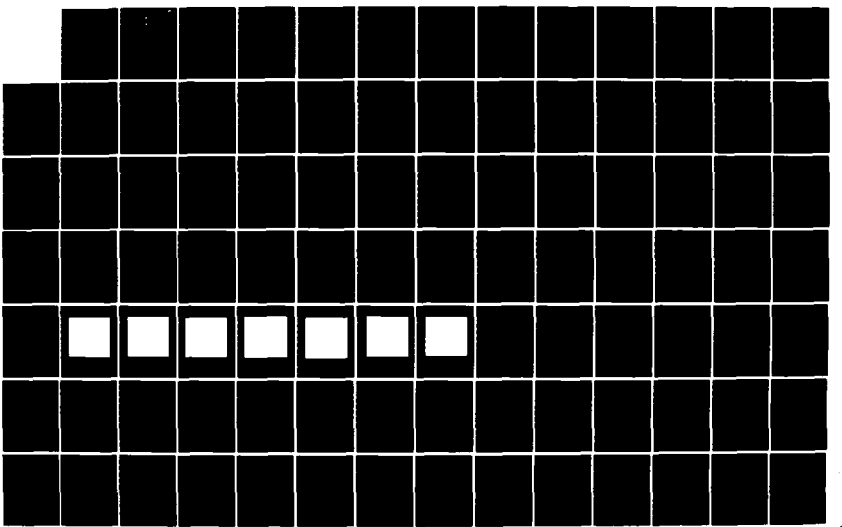
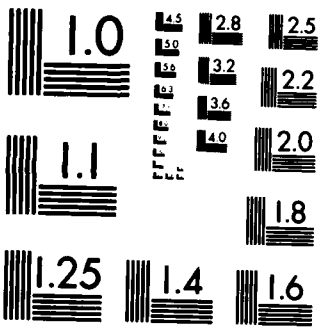
AD-A153 634 A GENERAL AREA AIR TRAFFIC CONTROLLER SIMULATION USING COLOUR GRAPHICS(U) DEFENCE RESEARCH ESTABLISHMENT OTTAWA (ONTARIO) B J FORD SEP 83 DREO-890 1/2

UNCLASSIFIED F/G 17/7 NL





National Defence / Défense nationale

# A GENERAL AREA AIR TRAFFIC CONTROLLER SIMULATION USING COLOUR GRAPHICS

by

Barbara J. Ford



"Original contains color plates: All DTIC reproductions will be in black and white"



DEFENCE RESEARCH ESTABLISHMENT OTTAWA
REPORT 890

Canada



September 1983
Ottawa

National Defence / Défense nationale

# A GENERAL AREA AIR TRAFFIC CONTROLLER SIMULATION USING COLOUR GRAPHICS

by

**Barbara J. Ford**
*Radar ESM Section*
*Electronic Warfare Division*



DEFENCE RESEARCH ESTABLISHMENT OTTAWA
REPORT 890

PCN
31B

September 1983
Ottawa

## ABSTRACT

The increased speed of aircraft and a greater density of air traffic are overtaxing air traffic controllers. Automation of some portions of the controller's information and the use of colour to highlight dangerous situations will make the controller's work easier and possibly more accurate. One of the major aims of the study is to show that the use of different colours to indicate varying distances between aircraft is a definite improvement over a monochrome display. The other major aim is to study the advantages and disadvantages of vector display and raster display technologies when applied to the air traffic controller scenario. A new set of graphics display routines was developed to be used for the simulation. Much consideration was given to the best methods to optimize the display routines, with attention given to real-time constraints.



## RÉSUMÉ

Les vitesses de plus en plus grandes des aéronefs et la densité croissante du trafic aérien imposent aux contrôleurs de la circulation aérienne une surcharge de travail. Ce travail devrait s'alléger, et peut être même devenir plus précis, grâce à une automatisation partielle des informations que reçoit le contrôleur ainsi qu'à l'emploi de couleurs pour faire ressortir les situations potentiellement dangereuses. L'emploi de couleurs pour indiquer les variations d'espacement entre aéronefs marque un progrès notable sur l'affichage monochrome, comme le démontre l'étude, dont c'est un des objectifs principaux. L'autre objectif, tout aussi important, est de mettre en relief les avantages et les inconvénients des technologies d'affichage à trame et vectoriel, dans le domaine du contrôle de la circulation aérienne. Un nouvel ensemble de sous-programmes d'affichage de données graphiques a été établi pour des fins de simulation. Une étude en profondeur a porté sur les méthodes qui donneront les meilleurs sous-programmes d'affichage, avec une attention particulière aux contraintes en temps réel.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF ACRONYMS/ABBREVIATIONS

| | |
|---|---|
| CRT | Cathode Ray Tube |
| DMA | Direct Memory Access |
| FL | Flight Level |
| MOS | Metal Oxide Semiconductor |
| NDB | Non-directional Beacon |
| RAM | Random Access Memory |
| RGS | Raster Graphics Subsystem |
| SRL | Systems Research Laboratories |
| TACAN | Tactical Air Navigation |
| TOA | Time Of Arrival |
| VDP | Visual Data Processor |
| VGS | Vector Graphics Subsystem |
| VHF | Very High Frequency |
| VOR | VHF Omnidirectional Range |
| VORTAC | VOR/TACAN |

# CHAPTER 1

# INTRODUCTION

## 1.1 Introduction

The increasing speed and density of air traffic is forcing aviation agencies to adjust procedures and equipment to prevent air traffic controllers from becoming overtaxed. The solution to the overtaxing problem is to automate some portion of the controller's task.

Radar is employed to display the aircraft in a given area. The movement of the aircraft is followed and the expected track is estimated. Potential aircraft collisions can only be detected by viewing the entire scene at once on the display. By interposing a computer, further information about the aircraft can be displayed, such as flight number, altitude, path and collision information. To further improve the benefits of the display, colour may be introduced to warn the air traffic controller of possible impending collisions. A simulation of a general area air traffic controller scenario was developed to examine the value of using colour to forewarn the controller of potential collisions.

In order to create an adequate simulation a graphics language was needed on the system that would allow the movement of the aircraft to approximate real-time. As the existing sets of routines were far too slow an entirely new package was written and is fully described in Chapter 2 (the Graphics Display Library). Considerable care was taken to create a set of routines that would be most effective for the air traffic controller scenario. Future applications may also readily employ these routines.

The general area air traffic controller simulation is described in Chapter 3; the equations involved in moving the aircraft on the display are presented in Chapter 4; and the effective real-time speeds of the aircraft are discussed in Chapter 5.

Information that is not as significant as the information on the main display must still be made available to the air traffic controller. It was necessary to devise a suitable secondary display which is presented in Chapter 6.

The air traffic controller communicates with the pilots of the aircraft in his jurisdiction so some interaction was

required to allow similar communication in the simulation. The pseudo-controller can input to the terminal suggested speed and bearing changes and the computer executes these changes as though the pilot followed the instructions precisely. This interaction is described in Chapter 7.

A monochrome display of the general area air traffic controller simulation is used for comparison with the simulation using colour. The comparison is in Chapter 8.

Two types of display CRTs, a raster display and a vector display, were available on which to implement the simulation. A comparison is made of the two display technologies in Chapter 9.

Consideration was given to possible areas for enhancement of the simulation for future applications. The enhancements are presented in Chapter 10. The conclusions of the study are presented in Chapter 11.

Using the new graphics display library of routines and the special purpose routines the simulation was implemented and studied. The full simulation program (atcint.c), listed in Appendix 2, displays the area background, moves the coloured aircraft across the screen, and allows air traffic controller interaction. Various sections of this program will be referred to and described throughout the report.

## 1.2 Equipment Used

The host machine is a DIGITAL PDP-11/34 computer with 124k words of MOS memory, a 67 megabyte disk drive, a number of terminals, and a NORPAK graphics system which consists of a Raster Graphics Subsystem (RGS) on a Systems Research Laboratories (SRL) colour monitor, a Vector Graphics Subsystem (VGS) on a Kratos colour monitor, and the associated Visual Data Processors (VDPs). The VDPs convert program instructions into formats recognizable by the monitors.

The VDP has memory for colours and blinking status. The memory is 1k by 1k by 4 bits. Three of the 4 bits are for colour so $2^3 = 8$ colours are possible. The fourth bit is for the blink status. The VDP includes a video lookup table which stores the representation of a large number of colours, however the memory still limits the choice of colours to be displayed to eight at a time.

The operating system is UNIX with C as the programming language [2], [23].

# CHAPTER 2

# GRAPHICS DISPLAY LIBRARY

## 2.1 Introduction

The graphics display library is a series of routines that provide a means of defining what will be on the colour display (The Visual Data Processor (VDP) controlling a Raster Graphics Subsystem (RGS) and a Vector Graphics Subsystem (VGS)). The routines are written in the C language running under UNIX on a PDP-11/34 computer. When a program to create a display is compiled, this library of display routines is appended to the compiled main program. This display program would have a number of calls to a subset of these library routines.

The system of display routines was developed to improve the speed of response time over the NORPAK-supplied routines which were originally written for TELIDON. These routines were very general and had no optimization for the high resolution VDP. Some of the more complex TELIDON routine displays (especially those involving arcs) took up to five minutes to draw completely. In order for the air traffic controller simulation to be effective the simulated aircraft must move in a realistic manner. Also, when interactive changes are input these changes must be effected within seconds. As the TELIDON routines could not react sufficiently, new routines were required.

The library of routines developed contains software that is less general purpose and more restricted than the TELIDON oriented software. The TELIDON software allows sections of display to be added by user interaction. The new graphics display library deals only with fixed format displays. The basic content of the display must be predefined so only those predefined components may be altered, and only their colour, location, blink, intensity may change. Due to these restrictions there is much less overhead processing required for each function with the new graphics display library. A display involves displaying a collection of groups. For the purposes of the air traffic controller simulation all information on the screen can easily be predefined. Once these components, or groups of information are defined all the subsequent changes are just minor changes to these groups. Changes involve colour or position modifications to aircraft.

The library of routines allows a large portion of the display contents to be predefined and display changes to be

made quickly. This is because the graphical definitions reside in the VDP, waiting for portions to be selected for display. This minimizes the amount of data flowing between the host processor and the VDP.

This chapter assumes some knowledge of the UNIX operating system and the C language. Syntax diagrams for using the graphics display library routines are given in Appendix 9.

## 2.2 The Phases

The graphics display library (gdlib.c) is divided into four distinct phases:

1) initialization phase
2) subpicture definition phase
3) group definition phase
4) display phase.

Programs accessing the graphics display library must acknowledge each of these phases by using some instructions from each of these phases. The descriptions of the phases that follow indicate the minimum number of instruction accesses allowed in each case. A group defines a distinct attribute on the display, for example a vector or some text. Group information may be changed or updated by edit commands. A subpicture is a combination of display attributes which are treated as a set. Subpictures are used if the same set of attributes are to be repeated throughout the display.

### 2.2.1 Initialization Phase

The initialization phase consists of a call to gdinit(). This routine initializes variables concerned with keeping track of which subpicture or group is being described, initializes the terminal and initializes the character spacing. It is in this phase that the user decides whether to use the RGS (unit 0) or the VGS (unit 1). The unit value defaults to 0 but may be set to 1 if desired.

### 2.2.2 Subpicture Definition Phase

The subpicture definition phase immediately follows the initialization phase. It is not necessary to have any subpictures defined but a call to gdsubend() to end the phase is required. There may be many subpictures defined; each one starts with a call to gdsubopn() to open the subpicture and

ends with a call to gdsubcls() to close the subpicture. Once subpictures have been defined they cannot be edited. A subpicture describes a picture element that will likely be repeated more than once in the display. Subpictures would be used mainly to define special symbols, (eg. the symbol for an aircraft).

The body of the subpicture consists of any combination of calls to gdcolr(), gdints(), gdblnk(), gdaset(), gdcset(), gdvect(), gdivec(), gdtext(). These routines are definition commands and will be described later. Calls to gdcolr() are not recommended on the VGS because considerable time is taken to switch colours. When the VDP displays the screen picture it sorts by colour then displays all of one colour at a time. However, colours in subpictures cannot be accessed to sort.

A subpicture is restricted to 1024, 16 bit words of data in total. This poses no practical limitation on the subpicture size. For example a subpicture could set up more than 500 vectors or more than 1000 characters. The call to gdsubend() is required whether there are any subpictures defined or not.

### 2.2.3 Group Definition Phase

The group definition phase immediately follows the subpicture definition phase. This phase is terminated by a call to gdgrpend(). At least one group must be defined. These groups contain the commands that describe what will be on the display.

A group definition consists of zero or more calls to gdcolr(), gdints(), gdblnk(), gdaset(), gdcset(); and one or more calls to any one of the three types of drawing commands: gdvect() or gdivec(), gdtext(), gdsubp(). A group can have one of the text, vector or subpicture drawing commands but not more than one type. The drawing commands gdvect() and gdivec() may be intermixed since they are both vector drawing commands even though the latter draws invisible vectors.

The calls to the definition commands simply reset the default attributes assigned to the group. When the gdgrpend() call is made the attributes are filled into the group header along with the drawing commands, and output.

### 2.2.4 Display Phase

The display phase immediately follows the group

definition phase. This phase is terminated by another gdinit(). If the calling program terminates without another gdinit() then the picture will remain displayed until the screen is accessed by another program.

This phase consists of calls to gdattach(), gdupdate(), gdcursor(), gdpoll(), gddisply() and calls to any of the edit group which may be made in any order, depending on application. Descriptions of all of these routines follow.

## 2.3 The Commands

The commands are the actual routines that will be called by the user's program to draw on the display. With this concise set of commands any type of display may be drawn, however the set is more specifically useful for line drawings rather than filled or solid drawings.

### 2.3.1 Control Commands

gdinit():
This routine sets up a line of communication with the VGS or the RGS. Variables, the terminal and character spacing are initialized.

gdsubopn():
This routine indicates that a subpicture definition will begin. The routine checks that the phase is correct to open a subpicture, and that there is not an unclosed subpicture. Once a subpicture is defined it cannot be edited. The routine keeps track of the number of subpictures.

gdsubcls():
This routine indicates that the current subpicture definition will end. The routine checks that the phase is correct to close a subpicture and that there is indeed an open subpicture to close. If so, the subpicture information is written to the VDP.

gdsubend()
This routine marks the end of the subpicture definition phase. The routine checks that all subpictures that have been opened have been closed. If not, an error is indicated. If the state is correct to end the subpicture phase it is changed to group phase and the group header information is set up so that the group definition phase may begin. Initially the

group colour is red, the intensity is full, the screen is not blinking, the drawing pointer is set to (0,0), and the small character set is in use.

gdgrpopn():
This routine is called to begin definition of a group. It checks that the phase is the group definition phase; and it checks that there is not an unclosed group. The routine keeps track of the number of groups defined. Each group may define text, or vectors (visible and invisible), or subpicture calls; but may not define a combination of these.

gdgrpcls():
This routine closes the definition of a group. The routine checks that the phase is correct to close a group definition and that there is indeed an open group to close. If so, the number of drawing commands in this group is recorded, and the group information is sent to the VDP.

gdgrpend():
This routine marks the end of the group definition phase. The routine checks that all groups that have been opened have been closed. The phase is checked and if it is correct (phase 3 - group definition phase) then the program may proceed to the display phase.

gdupdate():
This routine updates the display with the given list of groups. The groups are presented in an array along with the number of groups included. The information is sent to the VDP. If the program is not in the display phase an error is indicated.

gdpoll():
This routine polls for the co-ordinates of the cursor. The cursor is attached to the trackball. The x position and the y position of the cursor are each read into corresponding variables. If the program is not in the display phase at the time of the call the error is indicated.

gddisply():
This routine turns the display refresh on or off. If there are to be a series of edits to the display it may be desirable to turn the refresh off until all of the edits have been completed and then turn the refresh on again. If the program is not in the display phase at the time of the call an error is indicated.

gdcursor():

This routine turns the trackball cursor on or off. The program must be in the display phase.

gdattach():

A group defines the shape and size of the cursor, and in order to display the cursor that group must be selected for display through the last update command (gdupdate()). The routine attaches the trackball to the group defining the cursor. The gdaset() value in the defining group is the actual position of the cursor. The information is sent to the VDP. The program must be in the display phase.

### 2.3.2 Subpicture and Group Definition Drawing Commands

gdcolr():

This routine defines the subpicture colour if in the subpicture definition phase, or changes the colour value in the group header if in the group definition phase. If not in either of these phases an error is indicated. The possible colours are:

0 - red
1 - orange
2 - amber
3 - yellow
4 - green.

(Note: the RGS allows variations in blue and purple as well, but as the aim was to make the RGS and VGS work with the same library of routines, the RGS had to limit its colours to those of the VGS.)

gdints():

This routine defines the intensity of the information displayed on the screen if in the subpicture phase, or changes the intensity value in the group header if in the group definition phase. If not in either of these phases an error is indicated. The possible intensities are 0 to 17 octal with 17 being full intensity and 0 being invisible. The parameter passed to this routine should contain the intensity.

gdblnk():

This routine indicates whether or not the subpicture elements will blink if in the subpicture definition phase; or the routine changes the blink status in the group header if in the group definition phase. If not in either phase an error is indicated. The two states of blink are:

0 - do not blink

1 - blink.

gdaset():

This routine sets up the absolute x, y position from which the next information will be drawn. In the subpicture definition phase the x, y set is defined. In the group definition phase the x, y set is modified in the header. If in neither of the phases, an error is indicated. The values of x and y must be greater than or equal to 0, and less than or equal to 1023. Each group can only do one absolute position set. If a group does not do an absolute position set then the set from the immediately preceding group is used.

gdcset():

This routine defines the subpicture character set choice if in the subpicture definition phase, or modifies the character set choice if in the group definition phase. If in neither of these phases, an error is indicated. The character set choice is:

0 - small
1 - large.

gdvect():

This routine sets up a series of one or more relative vectors to be drawn one after the other. The array of vectors may be introduced in either a subpicture or a group. If the program is neither in the subpicture definition phase nor the group definition phase an error is indicated. The array of vectors and the size of the array must be given.

gdivec():

This routine sets up a series of one or more invisible relative vectors to be drawn one after another. The array of vectors may be introduced in either a subpicture or a group. If the program is currently neither in the subpicture definition phase nor the group definition phase an error is indicated. The array of vectors and the size of the array must be given. Invisible vectors may be used if one requires the x, y pointer position to be at another location on the screen but does not wish to start another group.

gdtext():

This routine defines a text string to be written on the display, in either a subpicture or a group. If the program is neither in the subpicture definition phase nor the group definition phase an error is indicated. The string of characters will be sent as

follows: gdtext("ABC", 3). The string and the length of the string (including blanks) must be indicated. Note: the string can also be given in an array.

gdsubp():
This routine indicates which subpictures will be accessed from the present group. If the program is not in the group definition phase an error is indicated. As subpictures were defined they were numbered, starting at 1. The subpictures to be accessed are to be indicated using these numbers which are introduced in the group as an array. The size of this array must also be given.

### 2.3.3 Group Edit Commands

After the screen has been displayed portions may need changing. This is done by editing the desired groups and then redisplaying the screen using the gdupdate() command. In each edit command the group to be edited must be indicated, along with the new value(s) of what is being edited. Groups were numbered from 1 as they were created, and are identified using this number. The edit type must also be given:

0 - if the template is to be edited
1 - if the template and the display are to be edited.

When the display is edited (edit type = 1) an automatic update of the display occurs.

gdedcolr():
This routine edits a group's colour.

gdedints():
This routine edits a group's intensity.

gdedblnk():
This routine edits whether or not the information in the group will blink.

gdedaset():
This routine edits the group's absolute x, y pointer position.

gdedcset():
This routine edits the group's choice of character size.

gdedvect():
This routine edits a number of vectors (0 or more) in

a given group. The vectors to be modified are presented in an array the size of which must be given. Each array must be of contiguous vectors, with the number of the first vector to be changed, given. There may be more than one call to gdedvect() for a given group since all vectors to be changed are not likely contiguous.

gdedivec():
This routine is like gdedvect() but the vectors are invisible vectors.

gdedtext():
This routine edits a series of contiguous characters of a text string. The number of the first character of the series must be given to the routine along with the number of characters in the series to be changed. Of course the new string of characters must also be given. Since the text edit is a one for one character replacement the new string must be the same length as the old string. There may be more than one gdedtext() call per group.

gdedsubp():
This routine edits the subpictures to be called from the group. The subpictures are presented in an array, the size of which must be given. The subpictures to be edited must be contiguous, and the number of the first subpicture of the set must be given. There may be more than one gdedsubp() call per group.

## 2.4 Special Purpose Routines

The following two routines are not part of the library itself but their functions are very basic and therefore the routines are appended to applications programs along with the graphics display library. The special purpose routines are written using the graphics display library routines.

### 2.4.1 Dashed Lines

Part of the background display for the air traffic controller scenario requires dashed lines. Several graphics software packages include a routine to do this but as the graphics display library consists of the more fundamental drawing commands the routine has been written separately. The routine may be of use to programs other than the air traffic controller simulation program so it was written as an external

subroutine.

The dashed routine, called gddash.c (Appendix 5), is passed the x and y coordinates of the position at which the line is to begin and of the position at which the line is to end. The dashed line is created by drawing a visible vector and then an invisible vector sequentially until the line is completed. The length of the dash is constant no matter what the slope of the line is. There are slight differences in the implementation of the basic routine depending on whether the line is to be of a horizontal nature ($-1 < \text{slope} < 1$) or of a vertical nature, or whether the line is to be drawn left to right or right to left if horizontal, or top to bottom or bottom to top if vertical.

### 2.4.2 Circles

The ability to draw circles is another general requirement that may be desired by other programs at a future date so the circle routine is separate from the air traffic controller simulation program.

The circle routine, called gdcirc.c (Appendix 6), is given the coordinates for the center of the desired circle, the radius of the circle (where a radius of 512 units is the largest possible circle), the number of vectors used to draw the circle (the more vectors the smoother the curve), and the number of vectors to draw before skipping a vector (if a dashed circle is desired). With this information the routine follows a loop that calculates and writes the vectors to the display, creating the circle.

## 2.5 Summary

The library routines are simple, straightforward and easy to use.

Although the display library itself does not include specialized routines such as the routine to draw dashed lines and the routine to draw a circle, using the routines in the library, it is a straightforward matter to write any required specialized routines. Once these routines are written they can be appended to the calling program or even added to the display library.

Consistency among the library routines was emphasized. For example all information for vectors, groups, strings, and subpictures must be presented as arrays. The processing

overhead has been minimized to allow programs using the library routines to run with as little delay as possible. It is beneficial that the same library of routines can be used to display on the raster CRT or the vector CRT, with the differences in manipulating the two CRTs transparent to the user. The library of routines provides an effective base for display depiction.

Appendix 1 is a complete listing of the library of display routines.

# CHAPTER 3

# THE AIR TRAFFIC CONTROLLER SCENARIO

## 3.1 General Area Air Traffic Controller (Jurisdiction and Duties)

The purpose of the air traffic controller is to ensure the safe, orderly and expeditious flow of air traffic so he must ensure that aircraft do not come close enough to each other to pose danger of collision. Secondary objectives of the air traffic controller include avoiding delays to aircraft, reducing fuel consumption and minimizing noise to people on the ground.

The goal of maintaining a safe flow of air traffic is achieved by preserving separation standards. The separation standards are: aircraft at the same horizontal level may not pass within three miles of one another; and aircraft separated by less than three miles horizontally must be at least 1000 feet apart vertically. If two aircraft come within the distance of separation standards they are within "airmiss" distance. Airmisses are recorded as they may be of interest if there are adverse effects on the flight crew or if any avoidance manoeuvre injures crew or passengers or damages the aircraft. [21]

Air traffic control consists of more than one task, carried out by different controllers. For one task, involving ground movement control, the controller is responsible for directing any traffic (aircraft, cars, fire engines, towing vehicles) using the runways and taxiways. For another task, involving approach control, the controller is concerned with the horizontal and vertical separation of the aircraft, and the flight paths which they should follow to approach the runway in the immediate vicinity of the airport. There are other types of controllers but only one more controller task will be discussed: the task considered for this study of general area air traffic control.

The colour graphics developed for the study are intended to assist this general area air traffic controller in his task to control all aircraft in a given area. Usually the general area air traffic controller views a 15 nautical mile radius, but his responsibilities are normally for a 10 nautical mile radius. He is situated at the center of his circle of responsibility, and all aircraft within the 10 nautical mile radius circle, at any altitude, must allow control by him. All

aircraft outside the 10 but inside the 15 nautical mile radius circle that are flying over 6000 feet must also allow control by the general area controller. Aircraft, usually small aircraft, flying under 6000 feet are in uncontrolled air space. They must follow the standard rules but only use the controller when they require assistance. These small aircraft must be visible to the air traffic controllers by radar so that other aircraft under control can be advised of their presence if necessary.

## 3.2 Background Display

In order to aid the general area air traffic controller a suitable display background is required. Figures 1 through 6 exhibit the background.

The background display contains the 10 and 15 nautical mile radius circles of interest to the air traffic controller. On the outer circle are the degrees relative to north that are used by the controller when determining the direction in which the aircraft is going. The beacons towards which aircraft fly are indicated. As the area depicted is the Ottawa area there is a TACAN (Tactical air navigation) beacon, a VORTAC (VHF Omnidirectional Range/TACAN) beacon, and two non-directional beacons (NDB) [15]. The NDB's, or omnidirectional transmitters, give directional information. The location of the beacons is known, the direction of its emitted signal can be determined, and therefore the aircraft bearing can be calculated. The TACAN is a military omnibearing and distance measurement system. Aircraft range can be determined from the pulse emitted from the beacon. VORTAC is the colocation of VOR and TACAN. VOR is similar to NDB so the bearing and range can be determined with the VORTAC. Aircraft fly along airways but in the Ottawa area the airways overlap considerably so the airway is depicted by a line down its center and by the dashed line as shown on the display.

In order to give the controller some reference to the positions of the aircraft the major terrain features are denoted. These include the Ottawa River, the Gatineau River, the Rideau River, and some of the Gatineau hills.

Air traffic controllers receive weather reports on a continual basis by radio transmissions from meteorologists. In order to assist the controller the more important weather fronts are included on the display. A very slow moving weather front has been indicated on the Ottawa area display along with the direction the front is moving. The weather front does not move during the simulation as they only travel from 10 to 25 miles per hour, depending on the type of weather mass, and

this would not be noticeable during the one or two minute simulation. The weather information is included to indicate the amount of clutter it adds to the display. Different patterns would represent different types of weather and these would be known to the air traffic controllers.

The runways and the control tower are, of course, included in the background display as they are very important to the air traffic controller.

The details included in the background are the most distinctive characteristics of the Ottawa area. It would not be difficult to include other items, for example, military test sites over which to avoid flying, or other airports in the area.

### 3.3 Opinions on the Use of Colour

The availability of colour graphics has led to questions of its use in air traffic control. Would colour graphics be cost effective? Would colour hinder the air traffic controller more than aid him? Would colour be redundant? What is the most effective use of colour? Several authors have commented on these questions.

Lucas [16] states that it is well known that the human eye can distinguish differences in colours more readily than differences in grey levels so colour should be considered.

Christ and Teichner [7] tested the effects of colour on visual search and identification performance. The results led to the opinion that colour is superior to size, brightness, and shape as unidimensional target features, but inferior to alphanumeric symbols, especially with increased task complexity. Unidimensional means differing from each other in terms of only one stimulus dimension; all other attributes of the targets are held constant. The reason colour seemed inferior to alphanumeric symbols is perhaps the fact that the users were familiar with the practice of identifying alphanumeric symbols. The general feeling of the users was that colour in displays is less monotonous, and that it produces less eye strain and fatigue. However, the authors felt colour interfered with the ability to discriminate other target dimensions in that colour distracts the observer so that he does not attend to other events on the screen.

Hopkin [12] suggests colour may be used to indicate:
1) climbing or descending or level aircraft,
2) aircraft's height band,
3) aircraft being handed over to another controller,

4) eastbound or westbound aircraft,
5) aircraft on potential collision courses,
6) supersonic or subsonic aircraft,
7) aircraft requiring co-ordination or liaison.

Hopkin feels none of these uses is obviously supreme, however none of them have been fully tested for effectiveness. He suggested that some of the demonstrated advantages of colour may come not from its value as a visual code but from enhanced motivation, attention and articulation associated with its pleasing appearance. Apparently air traffic controllers participating in the testing of a visual search task showed a significant improvement using colour instead of monochrome displays, by reducing search time, but not by increasing accuracy. It was recommended that colour as a non-redundant code should be tested in cluttered display conditions as perhaps then the benefits of colour would show up.

Hopkin also concluded that people often believe they work better with colour displays but this belief, being illusory, could also be dangerous because they may rely too heavily on colour. The air traffic controller may rely too heavily on the screen indicating problems in red, for example, and therefore miss a problem that did not show in red. The controller may scan the screen less frequently and become reliant on the red indicators.

If the colours chosen for screen items are not optimum then items in different colours may seem more disimilar than they are and items in the same colour may be equated more than they should be. Blinking information may be used instead of colour to attract the attention of the air traffic controller. Although blinking information could be found quickly it could prove irritating and distracting if it persisted for some time.

Hopkin concluded colour graphics is much more costly than monochrome graphics so the improved performance of the air traffic controllers must be significant before colour should be used. He forecasts studies will continue into the various uses of colour, and their cost effectiveness. There is an increasing availability of cheaper raster technology displays primarily due to decreasing prices for RAM.

### 3.4 The Use of Colour to Avoid Collisions

One of the most important areas in which to use colour is in averting potential aircraft collisions. The general area air traffic controller situation has been chosen to demonstrate the effectiveness of using colour to highlight potential danger. The colours available are green, red, orange, amber and yellow. The raster graphics subsystem has potential for

blue and purple but the vector graphics subsystem does not. As the two subsystems are to be treated the same to allow a comparison only the vector graphics subsystem colours are used. (When comparing the vector and raster subsystem it is acknowledged that the raster subsystem has the added advantage of additional colours.) Green is quite different from the other four colours so it has been chosen for the background. The aircraft are red, orange or amber. Yellow has been chosen for the runways and control tower, so they would stand out from the background. Yellow may be too close in colour to orange and amber but the choices were limited.

The air traffic controller views the display containing the position information of the aircraft and if he observes a potential collision he will notify the aircraft involved of a recommended new bearing and new speed to follow. The controller can then watch the effect of the change on the display. The photographs of the display shown in Figure 1 and Figure 3 show typical settings of the colour displays. The aircraft are depicted by a "+" associated with a flight number and a flight level. The flight number includes two letters indicating the airline and three numbers identifying the aircraft, for example "AC 123". The flight level is a number representing hundreds of feet, for example 80 would mean 8000 feet. The flight information is depicted in the same colour as the aircraft.

It was decided that the best way to use colour to make the air traffic controller aware of potential collisions was to display in red all aircraft within airmiss distance of another aircraft. Aircraft within two times airmiss distance of another aircraft would be displayed in orange as they may soon approach a dangerous separation. All aircraft not in any potential danger would be displayed in amber.

When determining the colour of the aircraft only the horizontal airmiss distance is considered. As the purpose of the study is to evaluate how well colour informs the air traffic controller of aircraft in potential danger it was decided to assume aircraft at any altitude were to be considered. There is the requirement to warn pilots of horizontal airmiss infractions even if the altitudes are different because the height information may be in error or the aircraft might be in the process of climbing or descending. Using red to indicate the aircraft in the most immediate danger was the obvious choice as air traffic controllers normally associate red with danger. Red is the colour most likely to attract their attention. Amber and orange are colours associated with caution. Amber contains less red than orange does so is not likely to attract the air traffic controller's attention as readily as orange. The varying amounts of red content in the colour of the aircraft imply correspondingly varying amounts of attention required.

As aircraft travel over the display they change to red, orange or amber as their distances from other aircraft change. Eight aircraft are depicted in one simulation. With only eight aircraft the air traffic controller can probably determine in advance which ones will turn red. However when twenty-four aircraft are depicted there are too many aircraft to scan so without the benefit of the aircraft turning red when in potential danger some possible airmiss situation might be overlooked. Even with only eight aircraft the occasional near miss may be overlooked so the addition of colour is a definite improvement. The analysis of the benefits of a colour display compared with a monochrome display is presented in Chapter 8.

## 3.5 Obtaining Position Data in Reality

In a real situation, rather than simulation, the locations of the aircraft are obtained from radar. A rotating antenna sends out pulses and measures the time for the pulse to return. On the radar screen this information is plotted as a distance from center. Some pulses returned may be false information if the pulses are returned from weather masses, ground clutter, or buildings. To minimize this false alarm rate the signal threshold should be set high enough to eliminate most of the false pulses but low enough to not block aircraft information.

The aircraft position data can be extracted from the radar screen manually with the advantage that false pulse returns can be detected and discarded by experienced operators. The disadvantages are the chance of human error, and the infeasibility of manually entering the data in the computer.

Plot extractors can also be used to retrieve the radar screen position data. This electronic method cannot detect all false alarms but human error is eliminated. A track initiator indicates the aircraft track or path and only aircraft would have a track so other pulse data can be eliminated. Once the radar aircraft position is known, the format of this data can be used by the computer and with minor calculations be transformed to information to be put on the display.

# CHAPTER 4

# CALCULATIONS FOR A MOVING DISPLAY

## 4.1 Path Calculations

After deciding on the needs of the general area air traffic controller the format of the background display was decided. Then the method of representing aircraft in the area and the best use of colour for the scenario was chosen. The next step was to show the aircraft moving. The position changes of the aircraft could not be random because in reality aircraft follow some sort of path. It was decided to have the aircraft each follow a straight line. The equations required to implement a series of straight lines are easily derived. Hyperbolic or elliptic paths were considered, but in the case of an aircraft passing through the area of the display the straight line is the best approximation of its route. In addition, more complex paths would require more time to produce a routine to implement them and the computer would use more execution time.

The equations used in determining the next positions of the aircraft will now be developed using the following variables:

yc = the current y value.
xc = the current x value.
yn = the new y value.
xn = the new x value.
p = the distance between the current coordinate and the new coordinate (in pixels or units).
m = the slope of the line of travel.

It is known that the distance between two points is:

$$(4.1) \quad p = \sqrt{(xc - xn)^2 + (yc - yn)^2}$$

The slope of a line is:

$$(4.2) \quad m = (yc - yn) / (xc - xn)$$

From this follows:

$$(4.3)\quad xc - xn = (yc / m) - (yn / m)$$

The following three equations lead to determining yn:

$$(4.4)\quad p^2 = ((-yn / m) + (yc / m))^2 + (yc - yn)^2$$

$$(4.5)\quad p^2 = yn^2 / m^2 - 2(yn)(yc) / m^2 + yc^2 / m^2 + yc^2 - 2(yn)(yc) + yn^2$$

$$(4.6)\quad 0 = yn^2(1 + 1 / m^2) + yn(-2(yc) - 2(yc) / m^2) + (yc^2 + yc^2 / m^2 - p^2)$$

From this use the quadratic formula to determine yn:

$$(4.7)\quad yn = (-b \pm \sqrt{b^2 - 4ac}) / 2a$$

Where:

$$(4.8) \quad a = 1 + 1 / m^2$$

$$(4.9) \quad b = -2(yc)a$$

$$(4.10) \quad c = (yc^2)a - p^2$$

Once yn has been determined xn can be calculated from (4.2):

$$(4.11) \quad xn = (yn - yc + m(yc)) / m$$

The (xn, yn) coordinate is the new position of the aircraft and each aircraft runs through a loop of 40 changes.

## 4.2 Proximity Calculations

Aircraft that are within airmiss distance of another aircraft are displayed in red, those within twice airmiss distance are orange, and all others are amber. Three miles of horizontal separation with respect to distance on the display screen is approximately 40 pixels (picture elements) on the raster display, or equivalent distance on the vector display. Although the vector display does not have pixels the screen has been treated as though it were of the size 1024 units by 1024 units. Therefore the size of, or a position on, the vector display may be treated the same as the raster display. So an aircraft position determined to be less than or equal to 40 pixels or units from another aircraft position on the screen is illustrated in red. The aircraft is displayed in orange if more than 40 pixels or units but less than or equal to 80 pixels or units from another aircraft. If the aircraft is not calculated to be orange or red then it is displayed amber. All possible pairs of aircraft are considered when determining the colour.

# CHAPTER 5

# RUNNING SPEED

## 5.1 Speed of Aircraft

The speeds at which the aircraft appear to be travelling are in fact dependent on the time taken to do the calculations for a moving display, execute the front-end graphics routines, and go through the Visual Data Processor (VDP). On the Raster Graphics Subsystem (RGS) the front-end graphics routines take more time as there is an added package to make the RGS operate like the Vector Graphics Subsystem (VGS). As it was not known before programming the simulation how much time would be spent in each of these phases it was not known in advance at what speeds the aircraft would appear to travel. A variable ("p") for each aircraft has a value indicating the number of pixels or units to be skipped with each aircraft position move. The "p" values were modified to make the speeds of the aircraft on the raster system appear realistic. Speeds of aircraft can be changed dynamically through interaction as described in Chapter 7.

The effective speeds of the aircraft on the RGS were calculated as follows. The time taken for all the aircraft to move 40 times was 103 seconds. The radii of the distance circles in the background display are known, giving the estimate that 6.3 inches represents 22.9 miles. This ratio is used to determine the distance in miles each aircraft travelled, knowing the number of inches it covered on the screen. 103 seconds is .0286 hours so dividing the miles by this value will give the speed in miles per hour. The speeds of the following six aircraft for reasonable "p" values are as follows:

AC 441 covered 5.0 inches which represent 18.2 miles
therefore travelled at 636 miles per hour.
EA 123 covered 7.6 inches which represent 27.6 miles
therefore travelled at 965 miles per hour.
GX 741 covered 8.1 inches which represent 29.4 miles
therefore travelled at 1028 miles per hour.
AC 164 covered 2.0 inches which represent 7.3 miles
therefore travelled at 255 miles per hour.
AC 111 covered 4.3 inches which represent 15.6 miles
therefore travelled at 545 miles per hour.

Note that GX 741 travelling at 1028 miles per hour is faster than a normal aircraft speed. This fast speed was included to allow a wide range of travelling speeds to be viewed.

Similarly the effective speeds of the aircraft on the

VGS can be calculated. The aircraft each moved the same distance and the same number of times as on the RGS, but the elapsed time was only 55 seconds (or .0153 hours). The elapsed time is less because the series of routines required to make the graphic instructions acceptable for the RGS are not required. The speed of the same six aircraft without altering the "p" values are as follows:

AC 441 travelled at 1191 miles per hour.
EA 123 travelled at 1804 miles per hour.
GX 741 travelled at 1922 miles per hour.
AC 164 travelled at 477 miles per hour.
AC 111 travelled at 1020 miles per hour.

The speeds are too fast to be realistic but the "p" variables that alter the effective speeds were not changed to allow a more accurate comparison between the vector and raster displays. If parallelism with the raster display was not necessary the "p" variables could easily have been altered to make the effective aircraft speeds on the vector display more realistic. If a real air traffic control situation, rather than a simulation, was in effect then the VGS would give a more accurate depiction than the RGS since position information received from radar, for example, could be displayed with less delay on the VGS. The RGS would always be displaying the true aircraft positions a few seconds late, since the extra time would be required to process the data through the front-end routines described in section 9.1.

## 5.2 Improved Running Speed by Other Methods

Delays in processing new aircraft position data could be the result of:

1) The mathematical calculations,
2) The front-end graphics routines (in the case of the RGS),
3) The VDP firmware routines.

In the event that delays caused by 1) might be significant, methods of obtaining future aircraft positions other than through calculations once the aircraft are ready to move, were implemented. The calculations were completed in advance and all future moves were stored in a database to be retrieved from disk when needed. As an alternative, should the disk retrieval time prove to be significant, the database of information for moving the aircraft was stored in an array in the simulation program. Neither of these approaches proved to be necessary as the limiting factor turned out to be the VDP firmware routines and the front-end graphics routines.

## 5.3 Timings

Since decreasing the time spent on the calculations to move the aircraft did not affect the effective speed of the aircraft, the time spent in the VDP and in the front-end graphics routines was evaluated.

The UNIX "time" command was used to time the moving aircraft program (atcint.c, Appendix 2) running on the RGS. No interaction was effected during the timing. The timings were as follows (minutes:seconds):

```
real ---  1:50.0
user ---     3.7
sys  ---    11.7
```

"Real" indicates the elapsed time; "user" indicates the time spent executing in user mode; "sys" indicates the time spent executing in system mode. The elapsed time measurement is of greatest interest.

To assess the portion of this total elapsed time that was spent in the VDP the moving aircraft program was run in a way such that all the output that would go to the RGS VDP to be processed for the raster display was discarded as it was received. The front-end routines (section 9.1) were still in use during this timing. The timings of the moving aircraft program run under these conditions were as follows:

```
real ---  1:08.0
user ---     3.6
sys  ---    12.2
```

Therefore approximately 42 seconds are spent waiting for software to execute in the VDP.

Finally to determine the time spent in the front-end routines (described in section 9.1) the moving aircraft program running on the RGS was timed as it discarded VDP output and bypassed the front-end routines. The resultant timings were as follows:

```
real ---     7.0
user ---     3.6
sys  ---     2.8
```

Without the VDP or the front-end routines there is, of course, no output to the colour display. However it is interesting to note that so much time is spent in each of the VDP and the front-end routines. The VDP takes 42 seconds of the program's running time and the front-end routines take another 61 seconds; the actual running of the atcint.c program is only 7 seconds.

The same "time" function determined the time required to run the moving aircraft program (atcint.c), with no interaction, for the VGS to be as follows:

```
real ---    55.0
user ---     4.0
sys  ---     6.2
```

Of course the front-end routines described in section 9.1 are not used for the VGS. Therefore the "real" time for running on the VGS should be approximately 61 seconds less than the "real" time for running on the RGS. Indeed 1:50.0 minus 55.0 is 55.0 seconds.

The above timings indicate why the general area air traffic controller simulation is shorter on the vector display than on the raster display. The time measurements noted above demonstrate the computational penalties involved in complex graphics, especially those involving raster systems.

# CHAPTER 6

# THE SECONDARY DISPLAY

## 6.1 The Secondary Display

In addition to the primary display depicting the moving aircraft, the air traffic controller may require supplementary information, but to include information such as destination, speed or type of aircraft would clutter the main display. For supplementary information, air traffic controllers currently use flight progress strips which are small strips of paper that have the extra flight information written on them. A more convenient technique is to use a secondary display that lists extra information about each aircraft. The secondary display can be displayed whenever the information is required so there is no need to fumble with bits of paper; and if changes in aircraft data occur the secondary display can easily be modified. A suitable secondary display has been presented and contains the following information: (Appendix 3 is a listing of the routine that draws the secondary display and Figure 7 shows the result.)

1) The flight number or id identifies the aircraft to which the adjacent information applies.

2) The flight level (FL) denotes the altitude at which the aircraft is flying, in hundreds of feet.

3) The cleared flight level is included if the aircraft has permission to change altitudes.

4) The type of aircraft indicates if the aircraft is a DC9, a Boeing 707, a DC10 or one of a number of other types.

5) The source denotes the airport at which the aircraft began its flight. These are represented by the airport codes, some of which are listed in Appendix 4.

6) The destination denotes the airport at which the aircraft is to end its flight, once again represented by the airport code.

7) The time of arrival (TOA) indicates the time the aircraft is to arrive at its destination.

8) The speed (in miles per hour) shows how fast the

aircraft is travelling.

As the secondary display is not the focus of the study the process of modifying the secondary display as changes in aircraft data arise, is not examined. The data on the secondary display that would be most likely to require modification is the aircraft speed. As well, the flight level, cleared flight level, aircraft destination, or time of arrival might change.

For the secondary display to have suitable updates the program controlling the movement of the aircraft on the primary display and the program displaying the secondary display would have to be linked together. Any updates to the secondary display would slow the movement of the aircraft. To make full use of the secondary display it should be displayed at the same time as the primary display. This is only possible if there are two functional vector or raster graphics subsystems available.

The secondary display has useful potential. The data on all aircraft are in the same place, on the display, but flight strips for the aircraft, spread across a table could become mixed up as they are used. The secondary display is readily available, information does not get mixed up or lost, and information on two or more aircraft can easily be compared because all information is grouped together.

# CHAPTER 7

# INTERACTION

## 7.1 The Interaction Scenario

The air traffic controller observes the aircraft and the paths they follow on the display. He can see when the possibility of a collision exists so interaction with the aircraft pilot is necessary to avert the collision. The air traffic controller then informs the pilot of his recommended new bearing, speed and, ideally, altitude. A method of simulating this controller/pilot interaction has been developed to further illustrate the effects of employing colour in a collision prevention scenario.

In this simulation the user acts as the air traffic controller. The movement of the aircraft is viewed on the colour graphics monitor and the proposed changes to the speed and bearing of an aircraft are entered on one of the terminals. The change that is entered will be implemented exactly and immediately. In reality the pilot would make the speed and bearing changes as fast as he can, allowing for aircraft reaction time.

To effect the change the user selects the aircraft, the new speed, and the new bearing. The format of input is as follows:

A B C

where A is the flight number of the aircraft (eg. AC_123),
B is the speed in miles per hour of the aircraft (positive integer),
C is the bearing of the aircraft (integer value between 0 and 359).

The bearing can be determined using the degrees marked on the outer circle of the display.

The most effective input medium is the terminal since a new speed and bearing must be entered. However, the actual selection of the aircraft to be redirected need not have been done by simply typing the flight number on the terminal. The aircraft could have been identified by a cursor controlled by the trackball, or if the equipment was available the aircraft could also have been identified by a light pen or by a "position indicator device". (A position indicator device allows the user to point to the displayed information and a matrix of fine luminous beams senses the location of the pointing

finger.) Typing the flight number on the terminal to identify the aircraft is, however, just as effective as these other methods and was the one adopted here for simplicity.

## 7.2 The Conversion from Input to Program Use

The input by the user, acting as the air traffic controller, is in terms familiar to him. The computer program controlling the movements of the aircraft (atcint.c - Appendix 2) does not immediately recognize the form of the input. The following conversions are necessary.

The flight number must be converted to a subscript (for eight aircraft the subscripts run from 1 to 8). In order to determine which subscript is applicable the program runs through a loop until the corresponding flight number with its associated subscript is found.

The speed of the aircraft is given in miles per hour but this must be converted to the number of pixels (on the raster display) or units (on the vector display) to skip with each move of the aircraft. A good approximation of the speed indicated by the movement on the raster screen is obtained by dividing the input speed by 30. For example an input speed of 450 miles per hour is 15 pixels or units.

The bearing must be converted to the slope of the line along which the aircraft travels and the direction (up or down) travelled along the line. In order to obtain the new slope the bearing must be converted to an appropriate degree value. The tangent of this degree value is the new slope. The bearing is taken from magnetic north which is 13 degrees west of true north. The degree value required for the tangent calculation begins at the bearing 103 degrees (ie. 90 degrees + 13 degrees) and goes counter clockwise. As a bearing is read in a clockwise direction a transformation is required:

> If ($13 \leq$ bearing $\leq 103$) then the degree value is (103 - bearing).
>
> If ($0 \leq$ bearing $\leq 12$) or ($104 \leq$ bearing $\leq 359$) then the degree value is (463 - bearing).

The slope of the path along which the aircraft travels is not sufficient information to determine the next position of the aircraft. The aircraft may be going up the line or it may be going down. This direction can be obtained from the bearing. If the bearing value is in the top half of the circle on the display describing the bearings then the direction is up; otherwise the direction is down. A quadratic equation is used to

determine the next position of the aircraft and the quadratic equation has a "+" or "-" in it. If the direction is up the "+" is used; if the direction is down the "-" is used. The top half of the circle is described by bearings between 0 and 103, and by bearings between 283 and 359, whereas the bottom half of the circle is described by bearings between 104 and 282.

With these converted values the quadratic equation to determine the next position of the aircraft has the data in the proper format.

## 7.3 Delay in Displaying Interactive Input

In order for the air traffic control simulation to be effective the interactive changes to aircraft speed or bearing must take effect without much delay. In a real situation it is not physically possible to have an aircraft change bearing instantly. Similarly a new speed is achieved only after some acceleration or deceleration. In addition to this the pilot takes some time to initially consider the suggestion, check his instruments and decide on the exact speed and course to follow. The resultant delay before a change has been completed is a few seconds.

The simulation on the RGS is somewhat realistic in that there is indeed some delay from the time the change is input to the time it is noted on the display. This delay is caused by the front-end routines described in section 9.1. There is a pipe (an inter-process communication channel) between the simulation routine and the front-end routines. Data going into the front-end routines is buffered since these routines cannot process the data fast enough. An interactive change is merely added to the buffered line up, therefore changes are not implemented immediately. This same phenomenon can be observed when the simulation program has finished running on the PDP-11/34 but the display is still changing.

The vector display does not have this front-end routine delay so almost as fast as the new speeds and bearings are input the changes are displayed. The speed with which the changes are effected is unrealistically fast. Delays could have been added to correct this but were not considered necessary for the comparison.

Since there is a delay on the raster system from the time the modifications are input, to the actual change on the screen, it might be necessary for the air traffic controller to initiate some path alterations when the aircraft symbols appear orange. They are orange if they are approaching an airmiss situation. The orange designation may be as important as the

red designation to the air traffic controller.

Even without the delay for interactive input to take effect there may not be enough time for the air traffic controller to react to the situation when aircraft symbols turn red. This would mean air traffic controllers should always be alert to orange. The problem with this is that orange and amber are too similar to attract sufficient attention to the orange. If they turned blue or purple they would likely be more noticable but unfortunately the vector display does not have these colours. Perhaps a solution would be to have aircraft appear red if less than twice airmiss distance from another aircraft and appear orange if less than four times airmiss distance but greater than twice airmiss distance.

## 7.4 System Routines Enabling Interaction

Interaction is possible between the user acting as air traffic controller at the terminal and the colour display. Interaction involves some code in the application program (atcint.c, Appendix 2), plus the system routines inread.c (Appendix 7) and ininit.c (Appendix 8). Very simply, the routine inread.c sends the lines of interactive input typed on the terminal, via a UNIX pipe to the application program which was passed as a parameter to inread.c. The routine inread.c notifies the application program that it has input by sending a signal 16. The routine ininit.c does the set up for the application program to receive the signal 16 from inread.c.

An application program uses the interactive input as it wishes. The routine atcint.c first checks that the input was valid, then divides up the input string so the correct portions are stored in the appropriate variables. Once these new values are in the proper variables the next access of these variables will cause the suggested change to take effect.

Using the system routines enabling interaction with the raster or vector displays to introduce interaction capabilities into the simulation, the reaction times of the air traffic controllers can be estimated. It is possible to determine the amount of time that passes before a controller recognizes a dangerous situation and suggests evasive action. Similarly it is possible to determine if the changes were implemented soon enough to avoid collision. These factors were discussed above.

# CHAPTER 8

# MONOCHROME VERSUS COLOUR

## 8.1 Description of the Monochrome Display

In order to fully appreciate the improvements achieved with the use of colour the program allowing interaction between a moving display and an air traffic controller has been implemented with all colour designations green. The result is a monochrome display that can be compared with the colour display. Since all other factors are exactly the same the effects of colour can be determined without any bias.

There are a number of methods that could be used to draw the attention of the air traffic controller to potential areas of collision; for example the aircraft in danger could have arrows pointing to them; or the alphanumeric characters and aircraft marker "+" representing the aircraft in danger could be displayed in special or large type. None of these other methods attracts the attention of the air traffic controller to the degree that the use of a distinctive colour does. These monochrome techniques have the disadvantage that the indicators of danger blend with the same coloured background and, therefore, are not readily visible. The air traffic controller will have to train himself to search for arrows or larger lettering as indicators of danger. It is clearly less tiring for the controller to simply scan the screen for aircraft displayed in a distinctive colour.

Another method to highlight the aircraft in danger on a monochrome display is to have them flash. The problem with this is that as all the aircraft move along their tracks they blink on and off and this is similar to flashing so it may confuse the air traffic controller.

No method of highlighting potential collisions on a monochrome display was considered to be of much advantage to the air traffic controller so the monochrome display chosen for comparison with the colour display is simply the same as the colour version with all aircraft displayed in green at all times.

## 8.2 The Comparison

The colour display is conclusively more effective than

the monochrome display as can be seen in Figures 1 and 3 compared with Figures 2 and 4. The colours attract the attention of the air traffic controller and this is beneficial for more than one reason. The aircraft information on the monochrome display has a tendency to blend with the background information, therefore the same information using colour is less likely to be overlooked. The more cluttered the background the more effective is the use of colour. Even the use of only one extra colour to allow differentiation between background and aircraft is a definite enhancement.

With the colour display the controller does not have to pass judgement on the distance between aircraft. Using the monochrome display the controller must continuously be calculating approximate distances between aircraft to determine those on which to concentrate. Having the computer calculate these distances and display the differences using colour relieves much of the pressure on the controller; he knows what area of the screen on which to concentrate. Using colour would therefore be less fatiguing for the controller. Also, less experienced air traffic controllers would find it easier to learn how to use the screen information with a colour display. The less experienced controller would likely make fewer mistakes while he becomes familiar with the various aircraft situations.

Since the choice of colours is limited to the few colours available on the vector system there is not sufficient difference between the depiction of aircraft within twice air-miss distance of other aircraft (orange) and the depiction of aircraft not in any potential danger (amber) to be really effective. Therefore the air traffic controller must still judge the shadings of the aircraft. If optimum, or obviously different colours were used the controller would not need to waste time wondering if the colour is orange or actually amber.

If colour displays were widely used by air traffic controllers then controllers who have colour deficient vision would be at a disadvantage. Red-green colour deficiency is the most common. A controller with this deficiency would lose the advantage of having his attention attracted by red aircraft. If red aircraft were also displayed at a higher intensity there could still be a slight advantage. The advantage of some colour to differentiate the aircraft from the background still remains.

It is obvious from the photographs shown in Figures 1 through 6 that recognizing aircraft in potential danger is easier using colour. Colour becomes more important as more aircraft fill the area. An example of twenty-four aircraft on the display is shown in Figures 5 and 6.

# CHAPTER 9

# RASTER REFRESH VERSUS VECTOR REFRESH DISPLAYS

## 9.1 The Software Package for the Raster System to Emulate the Vector System

The system contains a software package that allows the raster graphics subsystem (RGS) to emulate the vector graphics subsystem (VGS). For the purposes of the study the running of the general area air traffic controller simulation on both display systems was desired to be as similar as possible. This package allows the simulation program: atcint.c, that controls the simulation, to be run on either system with the differences in controlling the two systems being transparent to the user. With this capability the user can run the same simulation program on whichever display he prefers.

The raster and vector technologies are quite different. This is evident with the different methods of moving an object on the displays. A set of routines will be entered before a graphics instruction accesses the raster display. These routines set up the instruction so that it is recognizable by the raster display.

The selection of the RGS or the VGS is determined by a global variable called "gdunit". When "gdunit" is 1 output is sent directly to the VGS via the DMA interface. When "gdunit" is 0, output is directed to the emulating routines via a pipe. The output is then transformed into commands suitable to the RGS.

There is a resultant time penalty to manipulate the RGS display through the front-end emulating routines. The air traffic controller simulation that moves each aircraft 40 times takes double the time to complete when run on the RGS instead of the VGS. Any operator interaction takes effect a few seconds later than the same interaction on the VGS; and actual aircraft movement takes twice as long on the RGS as on the VGS.

## 9.2 The Raster Refresh Display

A raster refresh (or raster scan) system to display textual and graphical information uses a monitor similar to that employed in a standard home television set. Instead of the screen being refreshed from a video broadcast signal, it is refreshed from a pixel memory. The pixel memory is simply

memory containing the information on the colour and blink status of each pixel. The display screen is made up of pixels or dots arranged in horizontal rows. The system used for this study has good resolution since it has 1024 pixels per row by 1024 rows. Each pixel consists of one or more bits of data that represent the particular blink status or colour value for that point. The blink status is represented by one bit for each pixel and each of red, green and blue is represented by another bit for each pixel. Sixty times a second the complete display image is refreshed from the pixel memory.

Raster systems have various advantages and disadvantages over other types of systems. Raster systems are effective when complex, filled static pictures are to be produced. The pictures can be built up, point by point, to form a completed picture. There is no limit to the amount of information that may be presented because the refresh rate is not dependent on the information displayed. The data from the pixel memory is always transferred to the screen at a rate of 60 times a second (60 Hz) and this is not dependent on whether the pixels are to be displayed in black (invisible) or any other colour. This means there will be no flicker even with the most complex displays or solid areas.

There is potential for many colours provided there is adequate memory. As the RGS now stands eight colours are available (including black and white), however there is accommodation for up to seven colour memory boards, allowing 2**7 = 128 colours.

One disadvantage of raster systems is the amount of data that must be generated to create an image in pixel memory. Scan conversion is necessary, in other words the picture description instructions must be converted to information about how each pixel is to be represented. For example, to generate a line on a high resolution raster system of size 1024 by 1024 pixels may require the position of more than 1024 points or pixels to be computed. This may require too much time for image generation to be useful in many applications. The difference in timings between the raster and vector systems discussed in section 6.3 indicates this disadvantage. Raster displays of later technology can do scan conversion more quickly so the time taken to put a picture on the display is considerably decreased.

Another disadvantage of raster systems is the problem associated with modifying a generated image in pixel memory. To remove or move an item, the "delete" software must first erase and then if desired redraw the item. This means that it must be known what was originally on the screen at the particular spot that is to be erased so that it can be redrawn in the background colour. A further complication arises when

displayed items overlap. If an item to be erased overlaps another on the screen, the portion of the overlapped item would be erased along with the object that was meant to be erased. Unless it is feasible to redraw the erased portion of the item that was overlapped the display would have holes in it. This phenomenon is illustrated as the aircraft pass over the background leaving gaps in the background.

Unfortunately the high resolution monitors generally suffer from a lack of brightness which the RGS demonstrates. This is a disadvantage in that the display may need to be viewed in a darkened room in order to fully see the display.

A disadvantage peculiar to the RGS used for this study is the poorly formed alphanumeric characters.

## 9.3 The Vector Refresh Display

A vector refresh (or vector scan, or beam penetration) system produces the image by drawing vectors on the screen in the order they are given. To maintain the image on the screen it must be constantly refreshed. Vector refresh systems are well suited to dynamic displays of character and/or line images.

The need to constantly refresh the image on the screen can cause a problem when the image is quite complex. As the amount of information on the screen increases, so does the time required to refresh the image. This is because the image is drawn one vector at a time and the speed at which the image is displayed is dependent on the speed at which the vectors can be processed. If the refresh time is too great then the refresh rate will decrease and the image on the screen will begin to flicker. To avoid flicker, very complex images and filled areas must be avoided. Unfortunately, even the air traffic controller display is complex enough to cause some flicker on the VGS.

An advantage of the vector system is the ease and speed with which one can erase or move images. As this type of system is typically refreshed at a rate of sixty times a second (60 Hz), an image will fade from the screen and a new one will be drawn in approximately one sixtieth of a second. Very little processing is required to move an object because objects are generally represented by a set of instructions that describe the various attributes of the object such as position and intensity. To move such an object requires only that its position attribute be modified and it will automatically be in the new position after the next refresh (i.e. after one sixtieth of a second).

A disadvantage of the vector refresh system is that only five colours are available and some systems allow even fewer than five colours. The colours are displayed by a single beam striking the appropriate phosphor. The phosphors are layered across the face of the screen and the beam voltage is altered to allow the beam to penetrate to the correct phosphor layer.

The colour green is available but the green phosphor emits significantly more light when stimulated than those of the other colours, therefore green is distractingly brighter. This makes green a poor choice for the background information on the vector display, unless a lesser intensity is chosen. Since the same software is used on both the RGS and the VGS the intensity of green on the VGS remains greater. Therefore, the advantages of attracting the attention of the air traffic controller through the use of red and orange, is partially lost. This drawback could be partially overcome by dropping the intensity of the green phosphor through software for future simulations on the VGS.

Of the remaining colours red is distinctly red but the colours orange, amber and yellow are very similar to each other. Perhaps this display system should not have claimed to be a five colour system. Some vector systems may have better representations of these colours, although the problem is basically due to the limited number of layers of screen phosphor that can be employed.

Another disadvantage of some vector refresh systems is the existence of bright dots appearing where vectors end or start. This occurs if the intensity of the ends of the vectors is not carefully controlled. These unwanted bright dots preclude the use of intensity as a dimension of the display as they are likely to distract the user from observing items on the display that were meant to appear more intense.

The VGS used for the study has a peculiar stroke-generated character set. The characters are not smooth and some are not immediately recognizable. A minor difference with RGS is that VGS characters are defined from their center whereas the RGS characters are defined from their lower left corner. Therefore the positioning of the characters is slightly different: the VGS characters are to the left and down half a character compared with RGS characters.

Other differences worthy of note between the VGS and the RGS are as follows:

The VGS background is displayed on the screen instantly whereas the RGS background takes considerable time to display. The reason for this is the difference in

technologies. For example in order for the VGS to display a vector it needs the start point and a displacement and then the beam is produced; however for the RGS to display a vector each point along the line must be calculated and then displayed. The line point calculations are done in the VDP.

The VGS has a much smaller display area than the RGS.

Overall, the VGS is brighter than the RGS.

The VGS may flicker with complex pictures but the RGS does not. In fact processing of new aircraft positions further interferes with the VGS refresh rate.

Not an implicit problem of the RGS but resulting from the requirement to have the graphics display routines the same for the RGS and VGS, is the following difference. The interactive changes to aircraft bearing and speed are not redisplayed immediately on the RGS due to the requirement to pass through the set of software routines indicated in 9.1. The need for extra processing in the RGS means that the RGS will always have a delay compared with the VGS. On the VGS as soon as the change is input the display shows the change.

Neither the vector display system nor the raster display system is the obvious choice for the air traffic controller simulation as each has its advantages and disadvantages. The raster display systems are less expensive than comparable vector display systems so the trend is towards improving the raster technology.

# CHAPTER 10

# POSSIBLE FUTURE IMPROVEMENTS

## 10.1 Possible Future Improvements

If the subject of this study is to be continued the following refinements or additions may be considered. The enhancements go beyond the scope of the topic of study but they are worth noting.

A minor enhancement would be deciding upon and using an optimum set of colours to depict the distances between aircraft. To allow a greater choice of colours the raster system would have to be employed. For better contrast red should still imply potential collision aircraft, but yellow could indicate aircraft requiring less attention. With the background in green and with blue designating all aircraft not requiring attention, the entire focus would be on the brighter red and yellow colours.

The general area air traffic controller is not concerned with landing aircraft but the display could show aircraft approaching a landing and then disappearing from the display as they touch down. However, an aircraft approaching a runway may have to alter from its straight line path so the path equation would become more intricate.

It may be considered an enhancement to have the weather fronts depicted in the background display actually move. For the purposes of a simulation lasting only a few minutes this is not necessary. A cold weather front generally moves 20 to 25 miles per hour and a warm weather front generally moves 10 to 15 miles per hour. This movement is negligible on a simulation of a few minutes. Only the most important weather masses should be displayed or the display would appear too cluttered. Other weather information may be printed on the secondary display or may be obtained from reports transmitted by meteorologists on a continuous basis.

Currently there are programs that display eight aircraft and twenty-four aircraft. It would be more general to have a program that would display any number of aircraft. Of course this general program would require some input from the user to determine the desired number of aircraft, the initial position of each aircraft, and the speed and bearing of each aircraft. Before the aircraft are initially displayed the speed would be converted to pixels or units to skip; the

bearing would be converted to the slope of the line of travel and the direction along the line; and the initial colour of each aircraft would be determined. As this takes considerable time at the beginning of each run of the program, a standard package with the information about eight aircraft was written. Currently it is assumed that all aircraft involved in airmisses manage to avoid collision. Further simulations might consider the side effects of an actual collision.

Information about the trailing path of the aircraft may be required. These paths could be represented by dots depicting the last three calculated positions of the aircraft. The trailing path positions could have been stored so there would be no need to recalculate them.

It would be useful to have faster response to interactive input on the RGS. If the raster display were chosen for implementation then the software could be rewritten to optimize the raster functions. The aim would no longer be to have the raster and vector systems function with the same library of routines.

A fancy enhancement would be a filter function allowing selection of an altitude of interest or a geographic block of interest. This could be used by the air traffic controller if he wished to concentrate on a subset of the entire display. This, however, could be dangerous since the controller might then miss something not on the screen but still within his jurisdiction.

The air traffic controller can suggest alterations to the speed and bearing of the aircraft. Realistically he should also be able to suggest altitude changes. It is more complicated to edit the altitude than the speed or the bearing. However if this addition were to be implemented the flight number could be used to determine the group number with the altitude information and use the text edit command (gdedtext()) to update the altitude. The study is more concerned with detecting horizontal airmisses using the introduced colour. Altitude is a less important factor when deciding on airmiss situations because the altitude information may be in error or the aircraft may be changing altitude. An enhancement to the simulation would be to calculate airmisses according to altitude. Currently the altitude that is displayed is just text and not a variable with a value to compare against other aircraft altitudes. In order to include vertical airmiss calculations another variable, and calculations and comparisons associated with the variable, must be incorporated.

Several improvements could be included in a simulation, training or actual working system. The reactions and wishes of the controllers should be considered before such improvements

are initiated.

# CHAPTER 11

# CONCLUSIONS

## 11.1 Conclusions

A major goal of this study was to investigate the benefits of using colour in a general area air traffic controller scenario by simulating the scenario. The results demonstrate that colour is a very effective method of drawing the attention of the air traffic controller to possible collisions (Figures 1 and 3). When the aircraft are depicted in the same colour as the background they blend into the background (Figures 2 and 4). The more cluttered the background is the more important the use of colour in preventing the aircraft from being overlooked (Figures 5 and 6). There is less strain on the controller by having the computer calculate the distances between aircraft and indicate, using colour, which aircraft are worth examining. Colour relieves the monotony of a monochrome display.

The vector display has the capacity for only five colours. Using only these five colours the acceptable colour designations include green as the background and amber, orange and red as the aircraft colours; red being the colour to imply danger and orange or amber to imply less requirement for attention. The raster display has the capacity for more colours including purple and blue. Using these extra colours on a raster display it may be possible to find a more optimal set of colours to indicate distance between aircraft.

The other major goal of this study was to write a new set of display routines that would allow as small a processing delay as possible in order to perform the simulation effectively. The routines originally provided with the VDP systems were general purpose TELIDON oriented routines which were very slow. For the purposes of the study an appropriate set of display routines could be much more restrictive and therefore would have much less overhead processing. The resulting library of display routines allows the simulation to run with a minimum of general processing delay.

A secondary goal of this study was to compare the advantages and disadvantages of the raster and vector displays. The comparison will aid decisions in choosing which of the raster or vector displays is most effective for future projects. The VGS has a very limited choice of colours; the RGS allows more choice. The VGS display has an annoying flicker even if

the display is only slightly complex; the RGS has essentially no flicker. With the two preceding comparisons the RGS is the obvious choice; there is also the consideration that the RGS is less costly than the VGS. However if execution time is a major consideration then the VGS is the best choice. The time taken to put a picture on the RGS is much greater than the instantaneous display on the VGS. Even processing data to be displayed is slightly slower on the RGS. For example the interactive changes to aircraft bearing and speed take effect immediately on the VGS but take a few seconds to take effect on the RGS.

Due to the need for the front-end routines to allow the software for the RGS to emulate that of the VGS the air traffic controller simulation that moves each aircraft 40 times takes twice the time to complete when run on the raster system instead of on the vector system. For a project dealing exclusively with the RGS a new set of graphics display routines could be written to optimize the speed to display on the raster screen, however the raster system would still be somewhat slower than the vector system. This is due to the difference in technologies. Recent advancements in raster technology have overcome some of the problems causing slower display times.

By timing the stages of processing and displaying the air traffic controller simulation it was determined that very little time was required for position of aircraft calculations. Considerable time was spent in the VDP processing the information to be displayed. In the case of the raster system that much time again was spent in the front-end routines that allow the raster system software to emulate the vector system software. This extra processing time is significant.

A possible format of a secondary display was presented. The secondary display presents information to the air traffic controller that is supplementary to the information on the primary display. The information includes aircraft speed, source and destination airports, and time of arrival at destination. By replacing the currently used flight progress strips with the secondary display the information is more readily available and can be more easily modified.

Many thoughts on the application of colour to an air traffic controller scenario have been presented. The ideas introduced may be used directly to create an actual air traffic controller package or they may be applied to other similar scenarios that may also be improved with the addition of colour. The system chosen will depend on the operational needs, speed of reaction, colours required, formats to be displayed, and cost. The user must be involved in deciding the most appropriate parameters.

## REFERENCES

[1] Beling, G.J., Drake, P., Kellner, A.D., Spaargaren, G.C.A., "An Interactive Raster Graphics Terminal Application For Military Information Display", STC PP-174, SHAPE Technical Centre, July 1980.

[2] Bell Laboratories, "Documents For Use With the UNIX Time-sharing System", Sixth Edition.

[3] Bersh, P., Moses, F.L., Maisano, R.E., "Investigation of the Strength of Association Between Graphic Symbology and Military Information", Technical Paper 324, Research Institute for the Behavioral and Social Sciences, September 1978.

[4] Beyer, R., Schenk, H.D., Zietlow, E., "Investigations on the Readability and Interpretability of Electronic Displays: Investigations on the Effectiveness of Brightness Coding and Colour Coding of Display Elements", DLR-FB 71-57, Royal Aircraft Establishment Library Translation No. 1641, 1973.

[5] Buckley, E.P., House, K., Rood, R., "Development of a Performance Criterion for Air Traffic Control Personnel Research Through Air Traffic Control Simulation", Report No. FAA-RD-78-71, Federal Aviation Administration July 1978.

[6] Carter, R.G., "Visual Search With Color", NBDL-M005, Naval Biodynamics Laboratory, October 1980.

[7] Christ, R.E., Teichner, W.H., "Color Research for Visual Displays", AD-767 066, New Mexico State University, July 1973.

[8] Ford, R.L., "A Three-Dimensional Computer Model For Traffic in Off-Route Airspace", RSRE Memorandum No. 3250 Royal Signals & Radar Establishment, April 1980.

[9] Goff, R.C., Willoughby, W., Barboza, G., "Dissemination of Weather Information in the Air Traffic Control System", U.S. Department of Transportation, Report No. FAA-RD-81-47, FAA-CT-81-44, June 1981.

[10] Graham, R.D., "Display Technologies", Software Kinetics Ltd. Report, 1982.

[11] Hopkin, V.D., "An Appraisal of Real-Time Simulation in Air Traffic Control", Technology in Air Traffic Control Training and Simulation, Proceedings Volume II, February

1978, pp. 43-47.

[12] Hopkin, V.D., "Colour Displays in Air Traffic Control", London: Institution of Electrical Engineers Conference Publication No. 150, 1977, pp. 46-49.

[13] Innes, L., "The Man-computer Interface Problem in Terminal Automation", CATCA, Volume 2, 1975, pp. 11-13.

[14] Innes, L., McCann, C., "Graphic Display Interaction - Part II: Information Structure and Basic Functions", DCIEM Technical Report No. 78X5, January 1978.

[15] Kayton, M., Fried, W.R., "Avionics Navigation Systems", John Wiley & Sons, 1969.

[16] Lucas, J.A., "A High Speed Disc Memory and a Color Image Display for a Small Computer", Technical Report 133-20, U.S. Army Topographic Command, October 1970.

[17] Monroe, E.G., Basinger, J.D., Beardsley, H.W., "CIG Visual Simulation Considerations for Air Traffic Control Tower Operations Training", Technology in Air Traffic Control Training and Simulation, Proceedings Volume II, February 1978, pp. 89-97.

[18] Nawrocki, L.H., "Graphic Versus Tote Display of Information in a Simulated Tactical Operations System", Technical Research Note 243, Research Institute for the Behavioral and Social Sciences, June 1973.

[19] Newman, W.M. and Sproull, R.F., "Principles of Interactive Computer Graphics", McGraw-Hill, U.S.A., 1973.

[20] NORPAK Limited, "IDS-VGS Programming Guide", Number D2389.

[21] Peters, G., "Air Traffic Control - A Man-Machine System" Systems Behaviour Module 2, The Open University Press, 1973.

[22] Poulton, E.C., Edwards, R.S., "A Possible Use of Colour to Code Sonar Displays", OES 4/74, Medical Research Council, March 1974.

[23] Ritchie, D.M., Thompson, K., "UNIX Programmer's Manual" Sixth Edition, Bell Laboratories, 1975.

[24] Schmit, V.P., "A Selective Review of Relevant Aspects of Colour Perception for the Application of Colour to Sonar Displays", Tech. Memo FS 114, Royal Aircraft Establishment, March 1977.

[25] Sher, L.D., "Flight Simulator: Use of Spacegraph Display in an Instructor/Operator Station", AFHRL-TR-80-60, Air Force Human Resources Laboratory, Texas, July 1981.

[26] Transport Canada (Air), "Introduction to JETS", April 1973.

[27] Whitfield, D., "Real-Time Simulation Studies of a Concept for Conflict Resolution", Technology in Air Traffic Control Training and Simulation, Proceedings Volume II, February 1978, pp. 68-75.

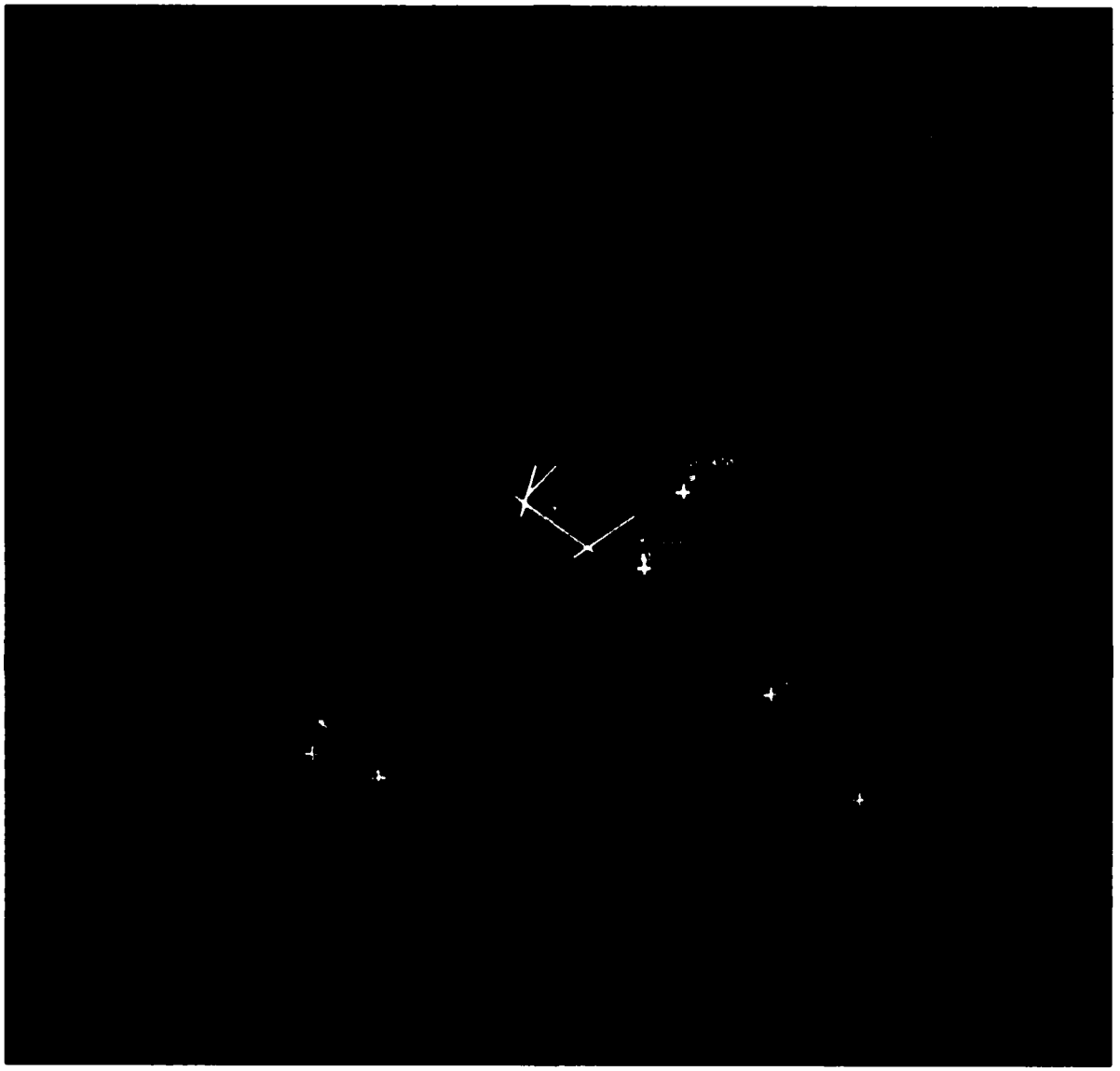


Fig.1: The eight aircraft air traffic controller colour display on the raster system.

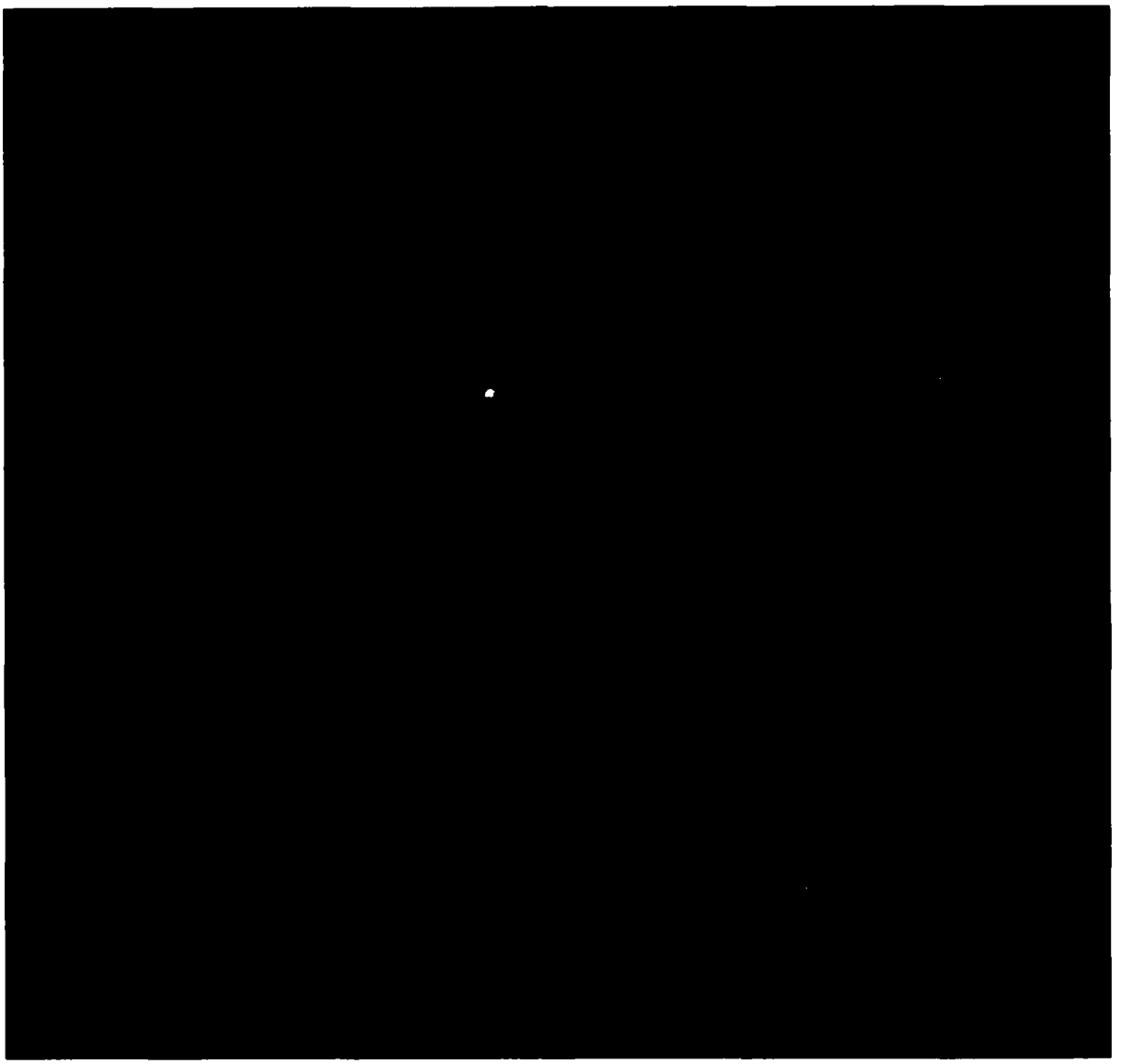


Fig. 2: The eight aircraft air traffic controller monochrome display on the raster system.

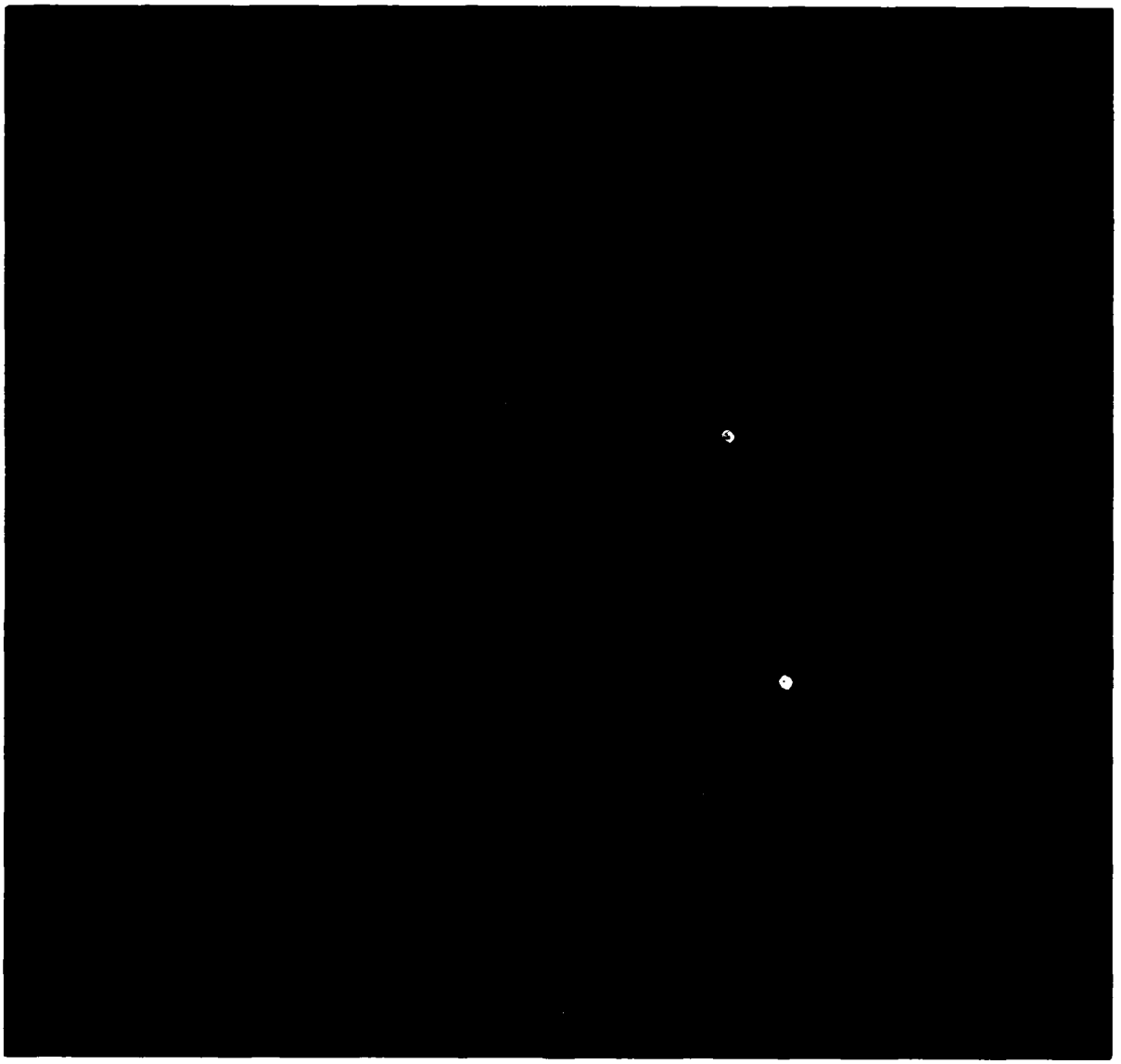


Fig. 3: The eight aircraft air traffic controller colour display on the vector system.

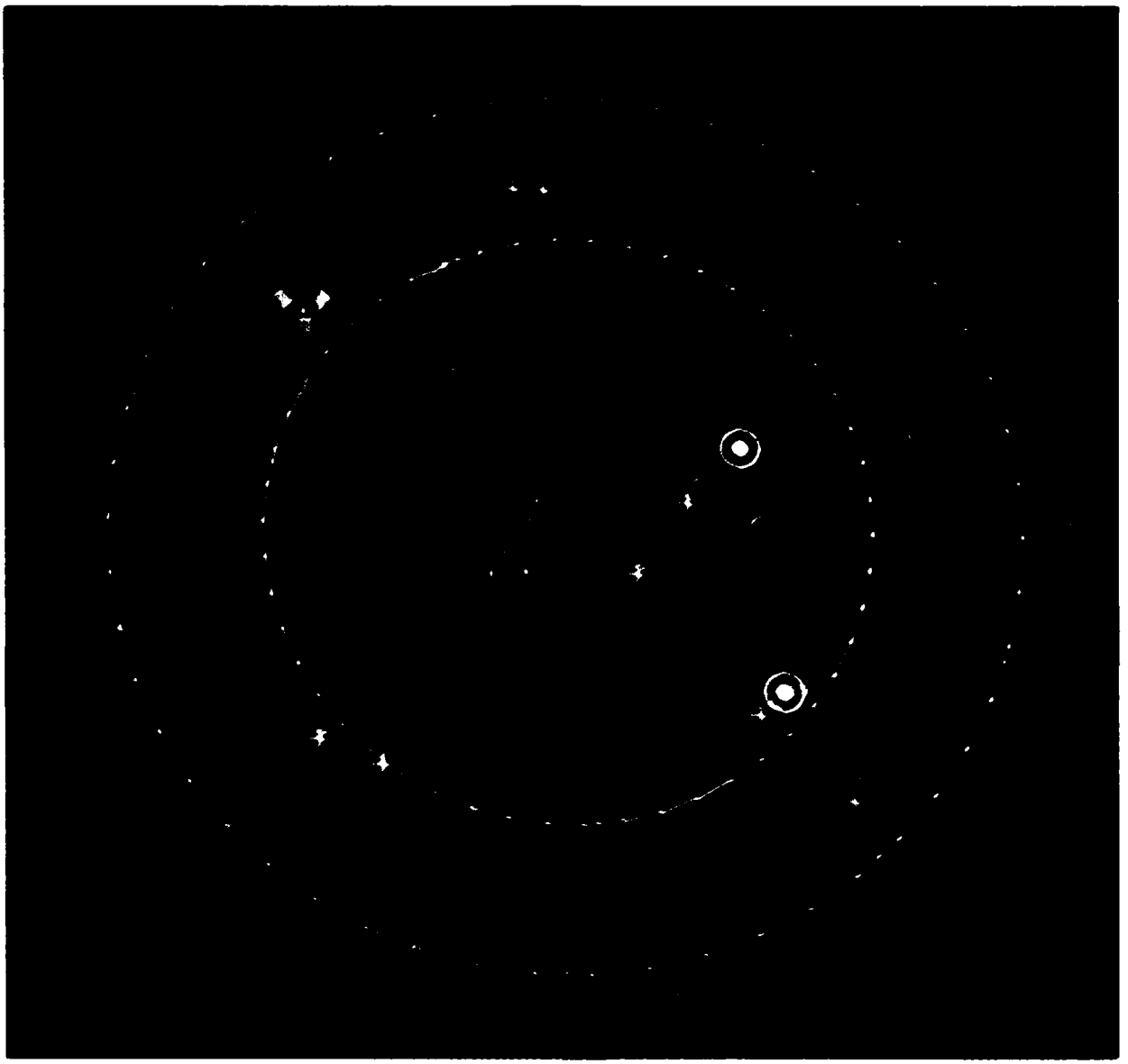


Fig. 4: The eight aircraft air traffic controller monochrome display on the vector system.

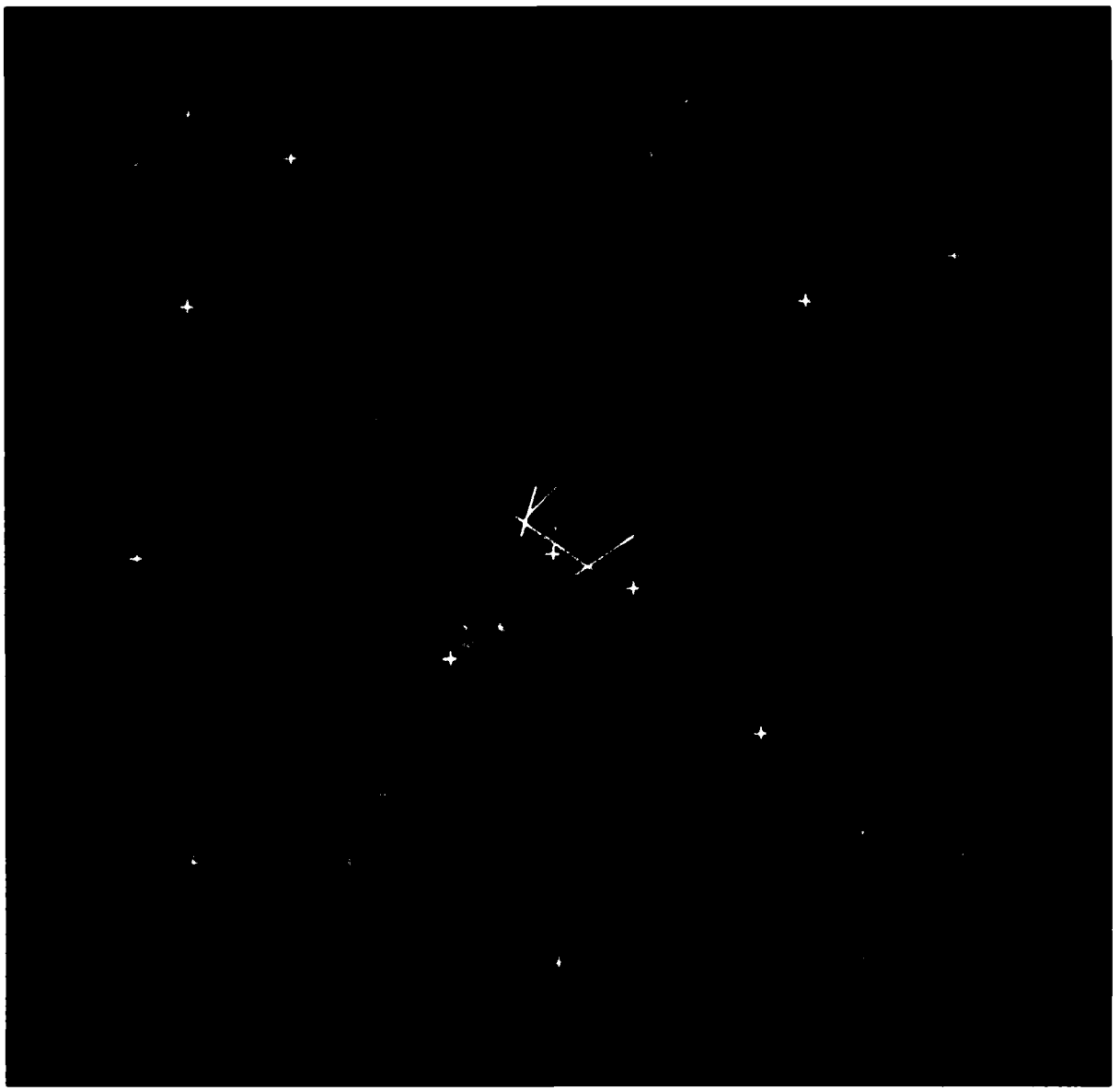


Fig. 5: The twenty-four aircraft air traffic controller colour display on the raster system.

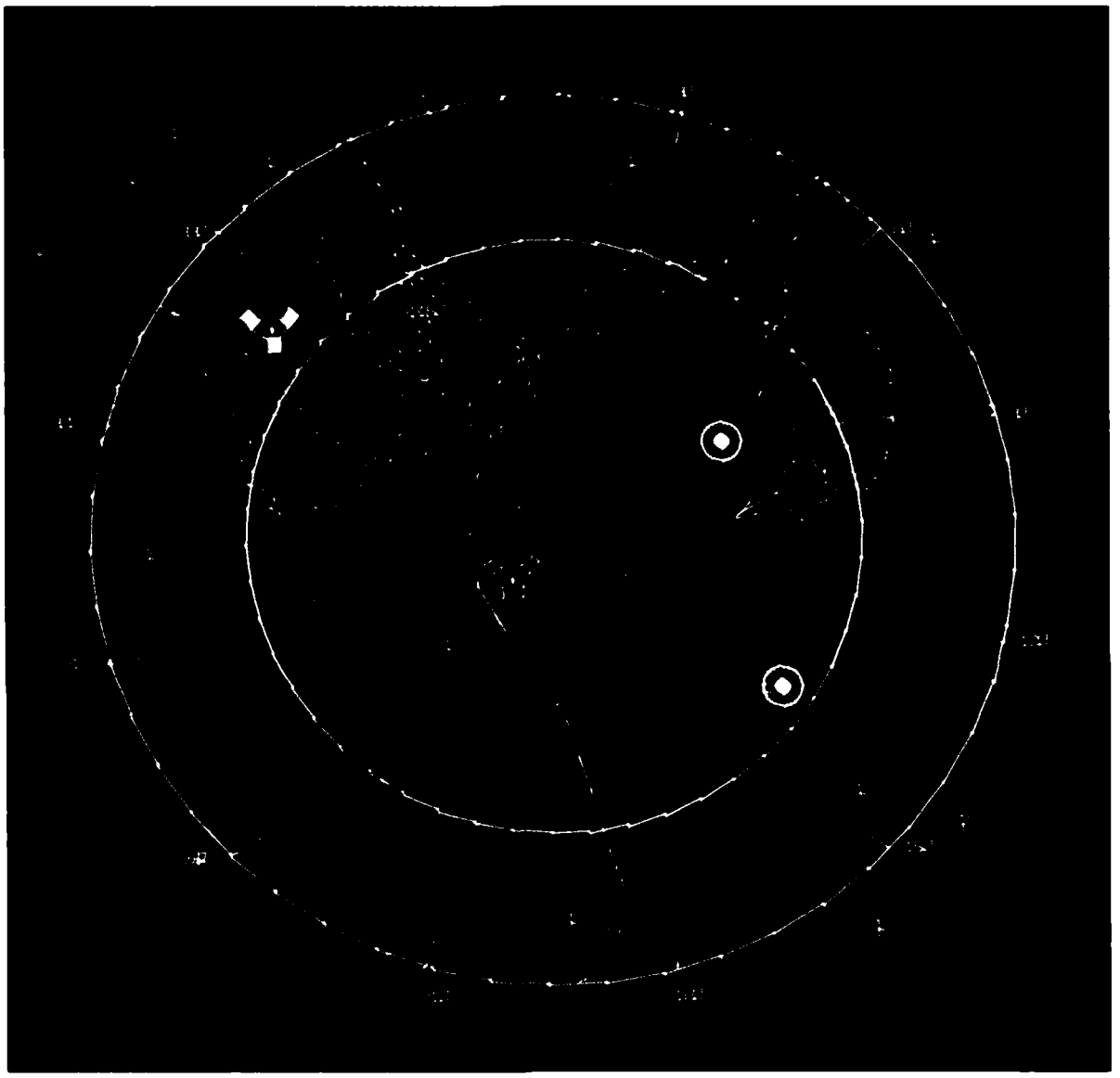


Fig. 6: The twenty-four aircraft air traffic controller colour displays on the vector system.

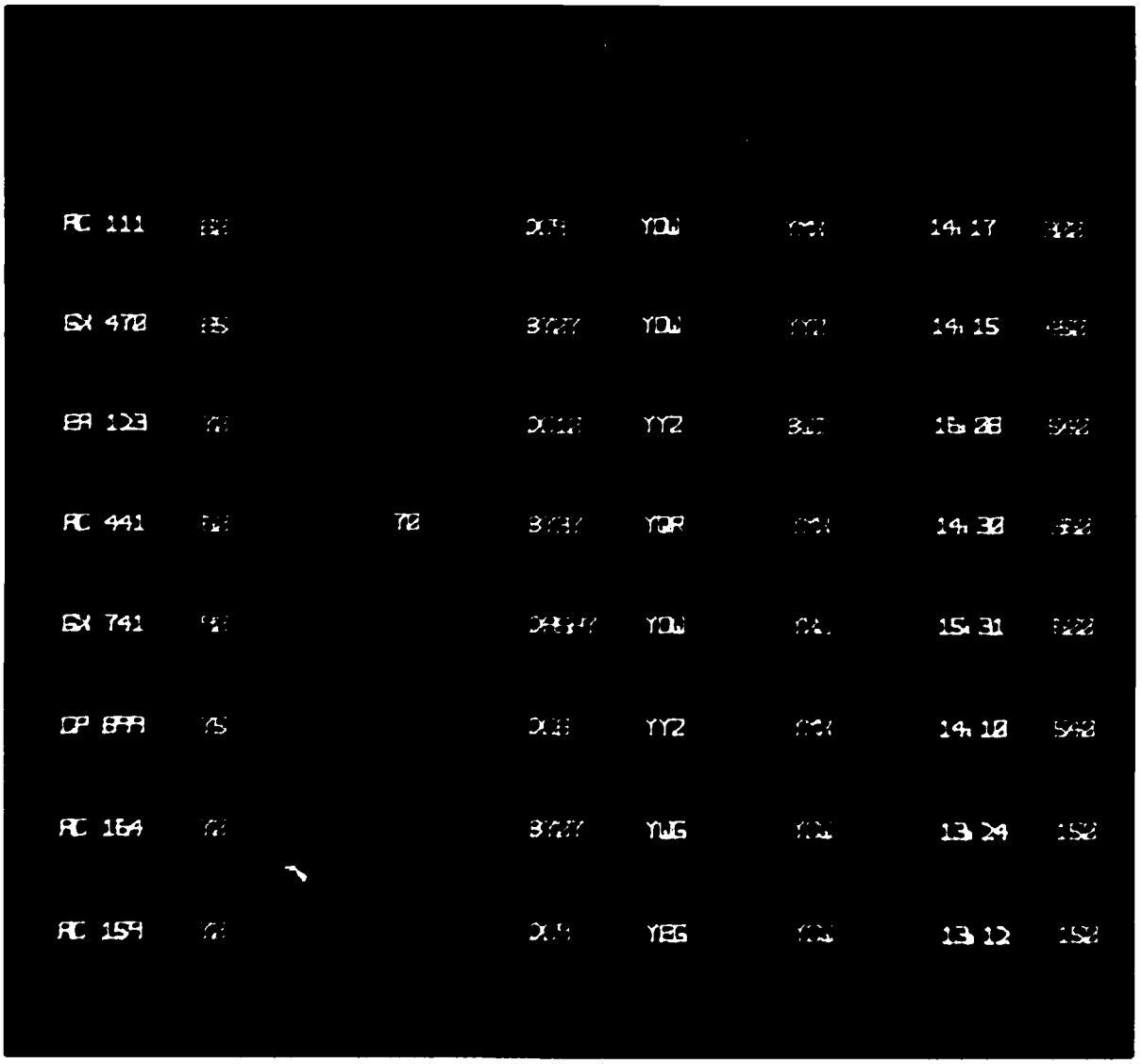



Fig. 7: The secondary display with information on eight aircraft.

# APPENDIX 1

## gdlib.c:
## THE LIBRARY OF DISPLAY ROUTINES

```
/* The global variables. */

int gdphase,              /* Keeps track of the phase:
                               1 - initialization.
                               2 - subpicture definition.
                               3 - group definition.
                               4 - display.                    */

    gdbuff[1024],         /* The buffer that stores all of the
                               subpicture group, header, and
                               display information before it is
                               sent to the VDP.                */

    *gdobptr,             /* The pointer into the output buffer
                               which stores information about the
                               group.                          */

    gdfid[2],             /* When a file is opened it is given
                               an id.                          */

    gddrcomcnt,           /* Stores the drawing command count
                               before it is put into the group
                               header.                         */

    gdspicno,             /* Indicates what subpicture number has
                               just been created. (Initially 1)*/

    gdspicph,             /* 0 - indicates all subpictures that
                                    had been opened have been
                                    closed.
                             1 - indicates there remains an open
                                    subpicture.               */

    gdgrpno,              /* Indicates what group number has just
                               been created. (Initially 1)     */

    gdgrpph,              /* 0 - indicates all groups that had
                                    been opened have been closed.
                             1 - indicates there remains an open
```

```
                                        group.                          */

     gdunit;                    /* Logical unit 0 - RGS, 1 - VGS.       */
                                /* Default is 0 - RGS.  If VGS is
                                     wanted do the following:
                                     gdunit = 1;
                                     gdinit();                          */

extern errno;                   /* external error number returned on a syscall.   */

/*****************************************************************************/


vwrite()

/* This routine sets up the write to the VDP. */

{    int bfindx;                /* This stores the size of the used part of the
                                     buffer.                                    */

     bfindx = gdobptr - gdbuff;
rewrit1:
     errno = 0;                 /* set the error number to 0. */
     if (gdunit == 0)
     {    if (write(gdfid[1], &bfindx,2) != 2)
          {    if(errno == 4) goto rewrit1;  /* ignore interrupted syscall. */
               printf("VDP write error.\n");
               exit(-1);
          }
     }
rewrit2:
     errno = 0;                 /* set the error number to 0. */
     if (write(gdfid[1], gdbuff, bfindx * 2) < 0)
     {    if(errno == 4) goto rewrit2;  /* ignore interrupted syscall. */
          printf("VDP write error.\n");
          exit(-1);
     }
     bfindx = 0;
     gdobptr = gdbuff;
```

```
}

/*****************************************************************************/

vread()

/* This routine sets up the read from the VDP. */

{    int  nbytes;

     if (gdunit == 0)
     {    *gdobptr++ = 0177777;
          vwrite();
     }
reread:
     errno = 0;          /* set the error number to 0. */
     if ( (nbytes = read(gdfid[0], gdbuff, 4)) != 4)
     {    if(errno == 4) goto reread;   /* ignore interrupted syscall. */
          printf("VDP read error : %d %d\n", errno, nbytes);
          exit(-1);
     }
}

/*****************************************************************************/

gdinit()

/* Please reference "NORPAK/ D2389/ IDS-VGS Programming Guide".

   This routine initializes variables and initializes terminal
     and character spacing.                                          */

{    static int flag;    /* Set on first program call to gdinit.  It keeps
                            track of number calls to gdinit.          */
     int forkid;         /* Id returned from fork call. */
```

```
char arg1[3];        /* Array for argumenttl. */
char arg2[3];        /* Array for argument 2. */
int  fid0[2];        /* Temporary fid array.  */
int  fid1[2];        /* Temporary fid array.  */

if (flag == 0)       /* First call to init. */
{    flag++;

     /* Rest of processing is dependant on unit. */

     if (gdunit == 0)          /* RGS. */
     {
          /* Set up a pipe to the IDSVDP program. */

          if (pipe(fid0) < 0)
          {    printf("Cannot create pipe.\n");
               exit(-1);
          }
          gdfid[0] = fid0[0]; /* get input fid.   */

          if (pipe(fid1) < 0)
          {    printf("Cannot create pipe.\n");
               exit(-1);
          }
          gdfid[1] = fid1[1]; /* get output fid.  */

          /* Time for a fork. */

          if ((forkid = fork()) < 0)
          {    printf("Cannot fork.\n");
               exit(-1);
          }

          if (forkid == 0)
          {
               /* We are the child process. */
               /* Create argument strings for the execute. */

               arg1[0] = fid1[0] / 10 + '0';
```

```
                arg1[1] = fid1[0] % 10 + '0';
                arg1[2] = 0;

                arg2[0] = fid0[1] / 10 + '0';
                arg2[1] = fid0[1] % 10 + '0';
                arg2[2] = 0;

                /* close excess fids.    */
                close(fid1[1]);
                close(fid0[0]);
                /* Must execute IDSVDP. */

                if (execl("/usr/bin/idsvdp","idsvdp",arg1,arg2,0) < 0)
                {    printf("execute failed\n");
                     exit(-1);
                }
           }
           else
           {
                /* Parent process. */
                sleep(2);

                /* close excess fids.    */
                close(fid1[0]);
                close(fid0[1]);
           }
      }
      else
      {
           /* Setup direct I/O to VGS. */

           if ((gdfid[0] = open("/dev/vgs",2)) < 0)
           {    printf("Cannot open VGS.\n");
                exit(-1);
           }
           gdfid[1] = gdfid[0];
      }
}
gdobptr = gdbuff;
```

```
        gdspicno = 0;
        gdspicph = 0;
        gdgrpno = 0;
        gdgrpph = 0;
        gdphase = 1;

        *gdobptr++ = 0100000;     /* Initialize terminal. */
        *gdobptr++ = 9;           /* Initialize character spacing. */
        *gdobptr++ = 13;
        vwrite();
}


/*************************************************************************/


gdsubopn()

/* This routine checks if the phase and state of the phase are right to
     open a subpicture.  If so gdspicno is incremented.                  */
/* Once a subpicture is defined it cannot be edited.                       */

{    if ((gdphase == 2) || (gdphase == 1))
     {    gdphase = 2;
          if (gdspicph == 0)
          {    gdspicph = 1;
               gdspicno =+ 1;
               gdbuff[0] = 0100002;
               gdobptr = &gdbuff[2];
          }
          else
          {    printf("There remains an open subpicture in attempt to open\n");
               printf("a subpicture %d.\n", gdspicno);
          }
     }
     else printf("Wrong phase when attempting to open a subpicture %d.\n",
                    gdspicno);
}
```

```
/*****************************************************************************/


gdsubcls()

/* This routine checks if the phase and state are right to close a
     subpicture.  If so the subpicture information is written to the VDP.  */

{    if (gdphase == 2)
     {    if (gdspicph == 1)
          {    gdspicph = 0;
               gdbuff[1] = (gdobptr - &gdbuff[2]);
               vwrite();
          }
          else
          {    printf("All subpictures already closed when attempting to\n");
               printf("close a subpicture %d.\n", gdspicno);
          }
     }
     else printf("Wrong phase to try to close a subpicture %d.\n", gdspicno);
}


/*****************************************************************************/


gdsubend()

/* This routine checks if the phase and state are right to end the
     subpicture phase.  If so the phase is changed to group phase and the
     group header information is set up.  This call must be included
     even if there are no subpictures.                                    */

{    if ((gdphase == 2) || (gdphase == 1))
     {    gdphase = 2;
          if (gdspicph == 0)
          {    gdphase = 3;
               gdbuff[2] = 0;           /* Red. */
```

```
                gdbuff[3] = 15;          /* Full intensity. */
                gdbuff[4] = 0;           /* Not blinking. */
                gdbuff[5] = 0;           /* x = 0 */
                gdbuff[6] = 0;           /* y = 0 */
                gdbuff[7] = 0;           /* First character set. */
           }
           else
           {    printf("There remains an open subpicture when trying to end\n");
                printf("the phase.\n");
           }
      }
      else printf("Wrong phase to end subpicture phase.\n");
}


/*************************************************************************/


gdgrpopn()

/* This routine checks if the phase and state are right to open a group.  If
     so the gdgrpno is incremented, the drawing command count is initialized,
     the first word of the buffer is set to the group opcode, and the buffer
     pointer is set to after the header information.                        */
/* NOTE:  Can only have text or vectors or subpictures in a group
     and not a combination.                                                 */

{    if (gdphase == 3)
     {    if (gdgrpph == 0)
          {    gdgrpno =+ 1;
               gdgrpph = 1;
               gddrcomcnt = 0;
               gdbuff[0] = 0100001;
          }
          else
          {    printf("There remains an open group in attempt to create\n");
               printf("a group %d.\n", gdgrpno);
          }
     }
```

```
     else printf("Wrong phase to try to open a group %d.\n", gdgrpno);
}


/*************************************************************************/


gdgrpcls()

/* This routine checks if the phase and state are right to close a group.
     If so the drawing command count is inserted into the group header
     and the group information is sent to the VDP.                        */

{    if (gdphase == 3)
     {    if (gdgrpph == 1)
          {    gdbuff[1] = gddrcomcnt;
               gdgrpph = 0;
               vwrite();
          }
          else
          {    printf("All groups already closed when attempting to close\n");
               printf("a group %d.\n", gdgrpno);
          }
     }
     else printf("Wrong phase to try to close a group %d.\n", gdgrpno);
}


/*************************************************************************/


gdgrpend()

/* This routine checks if the phase and state are right to end the group
     phase.  If so the phase is changed to display phase.                 */

{    if (gdphase == 3)
     {    if(gdgrpph == 0) gdphase = 4;
          else
```

```
        {    printf("There remains an open group when trying to end the\n");
             printf("phase.\n");
        }
    }
    else printf("Wrong phase to end group phase.\n");
}


/*************************************************************************/


gdupdate(grparay, grpcnt)

/* This routine indicates a new set of groups are to be displayed.  (Must
     be in phase 4.)  (This information is sent to the VDP.)              */

int grpcnt,          /* The size of the group array. */

    grparay[];       /* The array of new groups. */

{    int count;      /* A counter to go from 0 to the size of the group
                        array.                                           */

     if (gdphase == 4)
     {    *gdobptr++ = 0100004;
          *gdobptr++ = grpcnt;
          for (count = 0; count < grpcnt; count++)
               *gdobptr++ = grparay[count];
          vwrite();
     }
     else printf("Wrong phase to try gdupdate.\n");
}


/*************************************************************************/


gdpoll(x, y)
```

```
/* This routine reads the position of the cursor into the address of x and
     y.  (Must be in phase 4.)                                              */

int *x,         /* The x position of the cursor. */
    *y;         /* The y position of the cursor. */

{    if (gdphase == 4)
     {    vread();
          *x = gdbuff[0];
          *y = gdbuff[1];
     }
     else printf("Wrong phase to try gdpoll.\n");
}


/*************************************************************************/


gddisply(flag)

/* This routine turns the display refresh on or off.  This routine is only
     used for the VGS.  (Must be in phase 4.)                               */

int flag;       /* If flag = 0 then the display is refreshed.
                   If flag = 1 then the display refresh is inhibited. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100007;
          *gdobptr++ = flag;
          vwrite();
     }
     else printf("Wrong phase to try gddisply.\n");
}


/*************************************************************************/


gdcursor(flag)
```

```
/* This routine turns the cursor on or off.  When the cursor is on, the
     trackball is attached.  (The information is sent to the VDP.) (Must
     be in phase 4.)                                                    */

int flag;       /* If flag = 0 the cursor is off.
                   If flag = 1 the cursor is on.   */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100006;
          *gdobptr++ = flag;
          vwrite();
     }
     else printf("Wrong phase to try gdcursor.\n");
}


/*************************************************************************/


gdattach(groupno)

/* This routine attaches the trackball to the set position of the indicated
     group.  (The information is sent to the VDP.)  (Must be in phase 4.)  */

int groupno;   /* The indicated group number. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100003;
          *gdobptr++ = groupno;
          vwrite();
     }
     else printf("Wrong phase to try gdattach.\n");
}


/*************************************************************************/
```

```
gdcolr(colour)

/* This routine sets up the subpicture colour if in phase 2 or changes the
     colour in the group header if in phase 3.  The colours are:
     0 - red;  1 - orange;  2 - amber;  3 - yellow;  4 - green.          */

int colour;          /* The colour to be used. */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100020;
          *gdobptr++ = colour;
     }
     else if (gdphase == 3)
               gdbuff[2] = colour;
          else printf("Wrong phase to try gdcolr.\n");
}


/*************************************************************************/


gdints(intens)

/* This routine sets up the subpicture intensity if in phase 2 or changes
     the intensity in the group header if in phase 3.                    */

int intens;          /* The intensity to be used. (0 to 17 octal) */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100021;
          *gdobptr++ = intens;
     }
     else if (gdphase == 3)
               gdbuff[3] = intens;
          else printf("Wrong phase to try gdints.\n");
}


/*************************************************************************/
```

```
gdblnk(blnkst)

/* This routine sets up the blink status of the subpicture in phase 2 or
     changes the blink status in the group header if in phase 3.          */

int blnkst;          /* The blink status. (0 - off, 1 - on) */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100022;
          *gdobptr++ = blnkst;
     }
     else if (gdphase == 3)
               gdbuff[4] = blnkst;
          else printf("Wrong phase to try gdints.\n");
}


/*************************************************************************/


gdaset(absx, absy)

/* The absolute x, y position set is set up in the subpicture if in phase
     2 or changed in the header if in phase 3.                            */
/* NOTE:  Can only have one gdaset per group.                            */
/* NOTE:  If a group does not set an x and y position, the aset from the
          previous group is used.                                         */

int absx,            /* The absolute x position. */
    absy;            /* The absolute y position. */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100023;
          *gdobptr++ = absx;
          *gdobptr++ = absy;
     }
     else if (gdphase == 3)
```

```
          {    gdbuff[5] = absx;
               gdbuff[6] = absy;
          }
          else printf("Wrong phase to try gdaset.\n");
}


/*************************************************************************/


gdcset(charset)

/* This routine sets up the subpicture character set choice if in phase 2,
     or changes the character set choice in the group header if in phase 3.*/
/* 0 - small ;   1 - large .                                               */

int charset;         /* The character set choice. */

{    if (gdphase == 2)
     {    *gdobptr++ = 0100024;
          *gdobptr++ = charset;
     }
     else if (gdphase == 3)
               gdbuff[7] = charset;
          else printf("Wrong phase to try gdcset.\n");
}


/*************************************************************************/


gdvect(dxdyaray, nodxdy)

/* This routine sets up a series of vectors (in relative fashion) in either
     the subpicture or the group mode.                                     */

int dxdyaray[],      /* The array of relative points making up the vectors. */

     nodxdy;         /* The number of points in the array. */
```

```
{    int sizecnt,          /* This keeps track which dx or dy is currently
                              being handled.                                */

          temp;            /* This is used for intermediate calculations
                              on the dx or dy values.                       */

     if (gdphase == 2)
     {    *gdobptr++ = 0100030;
          sizecnt = 0;
          while (sizecnt <= (nodxdy * 2 - 1))
          {    temp = dxdyaray[sizecnt];
               temp = (temp + 1024) & 043777;      /* The dx, dy must be
                                                      positive, this
                                                      equation accomplishes
                                                      this.                 */

               *gdobptr++ = temp;
               sizecnt =+ 1;
          }
     }
     else if (gdphase == 3)
          {    if (gddrcomcnt == 0)
               {    gdobptr = gdbuff + 8;
                    *gdobptr++ = 0100030;
               }
               gddrcomcnt =+ nodxdy;
               sizecnt = 0;
               while (sizecnt <= (nodxdy * 2 - 1))
               {    temp = dxdyaray[sizecnt];
                    temp = (temp + 1024) & 2047;
                    *gdobptr++ = temp;
                    sizecnt =+ 1;
               }
          }
          else printf("Wrong phase when trying gdvect.\n");
}


/*****************************************************************************/
```

```
gdivec(dxdyaray, nodxdy)

/* This routine sets up a series of invisible relative vectors in either the
subpicture or the group mode.                                              */

int dxdyaray[],      /* The array of relative points making up the vectors. */
    nodxdy;          /* The number of points in the array.   */

{    int sizecnt,         /* This keeps track which dx or dy is currently
                             being handled.                                */

         temp;            /* This is used for intermediate calculations on
                             the dx or dy values.                          */

     if (gdphase == 2)
     {    sizecnt = 0;
          *gdobptr = 0100030;
          while (sizecnt <= (nodxdy * 2 - 1))
          {    temp = dxdyaray[sizecnt];
               temp = (temp + 1024) & 2047;       /* The dx, dy must be
                                                     positive, this
                                                     equation accomp-
                                                     lishes this.      */

               if ((sizecnt % 2) == 0)
                    temp = temp | 040000;         /* To indicate an invisible
                                                     vector set is des-
                                                     ired, bit 14 in the
                                                     dx data field is
                                                     set.              */

               *gdobptr++ = temp;
               sizecnt =+ 1;
          }
     }
     else if (gdphase == 3)
          {    if (gddrcomcnt == 0)
               {    gdobptr = gdbuff + 8;
                    *gdobptr++ = 0100030;
```

```
            }
            gddrcomcnt =+ nodxdy;
            sizecnt = 0;
            while (sizecnt <= (nodxdy * 2 - 1))
            {    temp = dxdyaray[sizecnt];
                 temp = (temp + 1024) & 2047;
                 if (( sizecnt % 2) == 0)
                      temp = temp | 040000;
                 *gdobptr++ = temp;
                 sizecnt =+ 1;
            }
       }
       else printf("Wrong phase when trying gdivec.\n");
}

/*************************************************************************/


gdtext(chstradr, count)

/* This routine sets up a series of characters in either the subpicture or
     the group mode.  The string will be sent as indicated: eg)
     gdtext("ABC", 3).                                                   */

int count;          /* This is the number of characters in the text string. */

char *chstradr;     /* This contains the text string. */

{    int tempcnt;   /* This is used to keep track of which character is
                       currently being handled.                          */

     if (gdphase == 2)
     {    *gdobptr++ = 0100031;
          for (tempcnt = 1; tempcnt <= count; tempcnt++)
          {    *gdobptr++ = *chstradr++;
          }
     }
     else if (gdphase == 3)
```

```
        {    if (gddrcomcnt == 0)
             {    gdobptr = gdbuff + 8;
                  *gdobptr++ = 0100031;
             }
             gddrcomcnt =+ count;
             for (tempcnt = 1; tempcnt <= count; tempcnt++)
             {    *gdobptr++ = *chstradr++;
             }
        }
        else printf("Wrong phase for gdtext.\n");
}


/*************************************************************************/


gdsubp(subparr, nosubp)

/* This routine will only allow phase 3.  It indicates a given subpicture
     will be called from the group currently being built.                 */

int subparr[],       /* This is the array of the subpicture numbers that
                             will be accessed from this group.           */
     nosubp;         /* This is the number of subpictures in the array.  */

{    int tempcnt;    /* This is used to keep track of which subpicture
                             entry is currently being handled.           */
     if (gdphase == 3)
     {    if (gddrcomcnt == 0)
          {    gdobptr = gdbuff + 8;          /* To avoid the header info.  */
               *gdobptr++ = 0100032;
          }
          gddrcomcnt =+ nosubp;
          for(tempcnt = 0; tempcnt < nosubp; tempcnt++)
               *gdobptr++ = subparr[tempcnt];
     }
     else printf("Wrong phase for gdsubp.\n");
}
```

```
/*************************************************************************/


gdedcolr(groupno, editype, colour)

/* This routine will only allow phase 4.  It edits the colour in a given
     group.                                                              */

int groupno,          /* The group number to be edited. */
     editype,         /* An indicator of whether the template (0) or both the
                         template and display (1) are to be edited.      */
                      /* Note:  when display is edited an automatic
                         update of the display occurs.                   */
     colour;          /* The new colour. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100020;
          *gdobptr++ = colour;
          vwrite();
     }
     else printf("Wrong phase to try gdedcolr.\n");
}


/*************************************************************************/


gdedints(groupno, editype, intens)

/* This routine will only allow phase 4.  It edits the intensity in a
     given group.                                                        */

int groupno,          /* The group to be edited. */

     editype,         /* An indicator of whether the template or both the
```

```
                        template and display are to be edited.       */
     intens;         /* The new intensity. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100021;
          *gdobptr++ = intens;
          vwrite();
     }
     else printf("Wrong phase to try gdedints.\n");
}


/*************************************************************************/


gdedblnk(groupno, editype, blnkst)

/* This routine will only allow phase 4.  It edits the blink status in a
     given group.                                                        */

int groupno,         /* The group to be edited. */

     editype,        /* Template or both template and display to be edited? */

     blnkst;         /* The new blink status. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100022;
          *gdobptr++ = blnkst;
          vwrite();
     }
     else printf("Wrong phase to try gdedblnk.\n");
```

```
}

/*************************************************************************/

gdedaset(groupno, editype, absx, absy)

/* This routine will only allow phase 4.  It edits the new absolute set of
     the given group.                                                      */

int groupno,          /* The group to be edited. */

     editype,         /* Template (0) or both template and display (1) to
                           be edited?                                      */

     absx,            /* The new absolute x position. */

     absy;            /* The new absolute y position. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100023;
          *gdobptr++ = absx;
          *gdobptr++ = absy;
          vwrite();
     }
     else printf("Wrong phase to try gdedaset.\n");
}

/*************************************************************************/

gdedcset(groupno, editype, charset)

/* This routine will only allow phase 4.  It edits the chosen character
```

```
     set of the given group.                                              */

int groupno,         /* The group to be edited. */

     editype,        /* Template (0) or both template and display (1) to
                        be edited?                                        */

     charset;        /* New character set choice. */

{    if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100024;
          *gdobptr++ = charset;
          vwrite();
     }
     else printf("Wrong phase to try gdedcset.\n");
}


/*************************************************************************/


gdedvect(groupno, editype, start, dxdyaray, nodrcom)

/* This routine will only allow phase 4.  The absolute vectors in a given
     group will be edited.  Since it is a drawing command opcode the start
     of the drawing command changes must be included (start);  the number
     of drawing commands to changed must be included (nodrcom);  and the
     array of new dx, dy values must be included (dxdyaray).              */

int groupno, editype, start, nodrcom, dxdyaray[];

{    int count,      /* This indicates which dx or dy we are currently
                        changing.                                         */

          temp;      /* This is used for intermediate calculations on the
                        dx or dy values.                                  */
```

```
     if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100030;
          *gdobptr++ = start;
          *gdobptr++ = nodrcom;
          for (count = 0; count <= (nodrcom * 2 - 1); count++)
          {    temp = dxdyaray[count];
               temp = (temp + 1024) & 2047;
               *gdobptr++ = temp;
          }
          vwrite();
     }
     else printf("Wrong phase to try gdedvect.\n");
}


/*************************************************************************/


gdedivec(groupno, editype, start, dxdyaray, nodrcom)

/* This routine will only allow phase 4.  The relative vectors in a given
     group will be edited.  Since it is a drawing command opcode the
     start of the drawing command changes must be included (start);  the
     number of drawing commands to be changed must be included (nodrcom);
     and the array of new dx, dy values must be included (dxdyaray).      */

int groupno, editype, start, nodrcom, dxdyaray[];

{    int count,     /* This indicates which dx or dy we are currently
                       changing.                                        */

          temp;     /* This is used for intermediate calculations on
                       the dx or dy values.                             */

     if (gdphase == 4)
```

```
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100030;
          *gdobptr++ = start;
          *gdobptr++ = nodrcom;
          for (count = 0; count <= (nodrcom * 2 - 1); count++)
          {    temp = dxdyaray[count];
               temp = (temp + 1024) & 2047;
               if ((count % 2) == 0)
                    temp = temp | 040000;
               *gdobptr++ = temp;
          }
          vwrite();
     }
     else printf("Wrong phase to try gdedivec.\n");
}


/*************************************************************************/


gdedtext(groupno, editype, start, chstradr, nodrcom)

/* This routine will only allow phase 4.  The text in a given group will be
     edited.  Since it is a drawing command there must:  the start of the
     drawing command changes (start);  the number of drawing commands
     (ie. characters to be changed) (nodrcom); and address of the character
     string (ie. new value) (chstradr).                                    */

int groupno, editype, start, nodrcom;
char *chstradr;

{    int count;          /* This keeps track of which character is
                            currently being handled.          */

     if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
```

```
          *gdobptr++ = editype;
          *gdobptr++ = 0100031;
          *gdobptr++ = start;
          *gdobptr++ = nodrcom;
          for (count = 1; count <= nodrcom; count++)
          {    *gdobptr++ = *chstradr++;
          }
          vwrite();
     }
     else printf("Wrong phase to try gdedtext.\n");
}


/*************************************************************************/


gdedsubp(groupno, editype, start, subparay, nodrcom)

/* This routine will only allow phase 4.  The subpicture number to be
     called from the given group will be edited.  Since it is a drawing
     command there must be:  The start of the drawing command changes
     (start);  the number of drawing commands or in this case the
     number of consecutive subpicture calls to be changed (nodrcom);
     and the array of new consecutive subpicture calls (subparay).    */

int groupno, editype, start, nodrcom, subparay[];

{    int count;          /* This keeps track of which subpicture call
                            is currently being changed.              */

     if (gdphase == 4)
     {    *gdobptr++ = 0100005;
          *gdobptr++ = groupno;
          *gdobptr++ = editype;
          *gdobptr++ = 0100032;
          *gdobptr++ = start;
          *gdobptr++ = nodrcom;
          for (count = 0; count <= nodrcom - 1; count++)
               *gdobptr++ = subparay[count];
```

```
            vwrite();
        }
        else printf("Wrong phase to try gdedsubp.\n");
}
```

# APPENDIX 2

## atcint.c:
## THE BASIC ROUTINE TO SIMULATE THE GENERAL AREA AIR TRAFFIC CONTROLLER SCENARIO AND TO ALLOW INTERACTION

```
/* The following does the general area air traffic controller simulation.
     The background depicting the Ottawa area is drawn.  Then eight
     aircraft are shown travelling across the screen.  The aircraft
     change colour as their proximities to other aircraft change.
     It is possible for the user to interact with the aircraft to
     change their bearing or speed.                                  */


zzvec(x, y)

/* This routine puts the next relative position in the format used by gdlib. */

int x, y;
{    int xy[2];
     xy[0] = x;
     xy[1] = y;
     gdvect(xy, 1);
}

zzivec(x, y)

/* This routine puts the next relative position for an invisible vector in the
     format used by gdlib.                                                    */

int x, y;
{    int xy[2];
     xy[0] = x;
     xy[1] = y;
     gdivec(xy, 1);
}

plane(xcentre, ycentre)

/* This routine draws the + that represents the aircraft. */

int xcentre, ycentre;    /* These are the coordinates for the centre of the +. */
{
     gdaset(xcentre - 7, ycentre - 1);
     zzvec(14, 0);
```

```
        zzivec(-14, 1);
        zzvec(14, 0);
        zzivec(-14, 1);
        zzvec(14, 0);
        zzivec(-7, 7);
        zzvec(0, -14);
        zzivec(1, 14);
        zzvec(0, -14);
        zzivec(1, 14);
        zzvec(0, -14);
}

compare(str1, str2)

/* This routine compares the characters in the strings. */

char *str1, *str2;

{       int count,
            char1, char2;  /* These are used to store one character at a
                              time of the strings.                        */

        for (count = 1; count <= 6; count++)
        {   if ((char1 = *str1++) != (char2 = *str2++)) return(0);
        }
        return(1);
}

extern gdunit;

main(argc, argv)

int argc;
char *argv[];

{       float pow();        /* pow(x,y):  x**y */

        double sin(), cos();
```

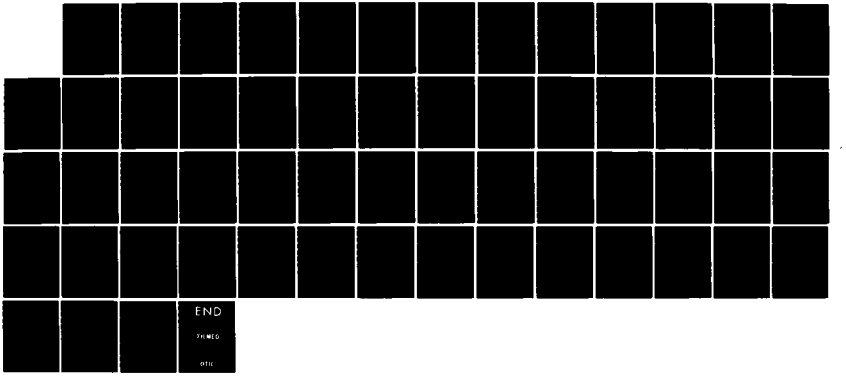
```
char fltno[6];          /* This stores the string indicating the
                           flight number.                          */

float radian;           /* sin and cos must be in radians. */

float spf;              /* A floating value of speed. */

int tempx, tempy;       /* These are used in drawing the beacons. */

int count;              /* This is used as a counter in for loops. */

int garray[119];        /* This is the array of group numbers to be displayed. */

int flag;               /* This indicates if there has been some interaction. */

int nbytes;             /* This is used in deciding if the interaction input
                           was valid.                              */

int subscr;             /* Each aircraft is referred to by a number, this
                           stores that number once it has been calculated. */

int speed;              /* This stores the aircraft speed to be input by
                           the atc when he wishes a change.  The speed
                           would reasonably be between 0 and 700 mph.   */

int bearing;            /* This stores the aircraft bearing to be input
                           by the atc when he wishes a change.  The
                           bearing should be between 0 and 359 degrees
                           taken from the 0 shown on the display.       */

int degree;             /* Bearing is translated to a degree value for
                           calculation purposes.                        */

char buf[70];           /* This will store the string printed on the
                           terminal indicating the changes to the
                           atc scenario.                                */

float x1, x2, y1, y2;     /* Temporary stores for xn and yn values
                             needed due to integer overflow problems. */
```

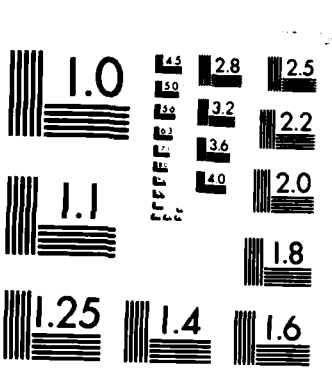
```
int xc[9];          /* The current x value of the plane position. */

int yc[9];          /* The current y value of the plane position. */

int p[9];           /* The number of points the plane moves (relative
                       to speed.                                    */

int xn[9];          /* The new calculated x value of the plane
                       position.                                    */

int yn[9];          /* The new calculated y value of the plane
                       position.                                    */

int move;           /* Counts the number of moves the planes make. */

int plncnt;         /* Counts through the number of planes in loops. */

int inner;          /* Used for an inner loop count. */

float dist;         /* The distance between pairs of new plane
                       positions.                                   */

int colour[9];      /* The colours of each of the planes. */

int group;          /* An indicator of the group being edited. */

int plormin[9];     /* Plus or minus in the quadratic equation to
                       calculate yn.                                */

float m[9];         /* The slope of the line the plane travels. */

float xnfloat, ynfloat;  /* Temporary stores for the new x and y
                            values of the plane position, respectively.  */

float a, b, c;      /* Temporary variables used in quadratic formula. */

if (argc > 1) gdunit = 1;          /* Set for VGS. */
ininit(&flag);      /* Initialize for interaction. */
```

```
gdinit();
gdsubend();

/* Distance circles. */

gdgrpopn();
gdcolr(4);
gdcirc(512, 512, 300, 50, 300);
gdgrpcls();
gdgrpopn();
gdcirc(512, 512, 301, 50, 300);
gdgrpcls();
gdgrpopn();
gdcirc(512, 512, 302, 50, 300);
gdgrpcls();
gdgrpopn();
gdcirc(512, 512, 450, 50, 300);
gdgrpcls();
gdgrpopn();
gdcirc(512, 512, 451, 50, 300);
gdgrpcls();
gdgrpopn();
gdcirc(512, 512, 452, 50, 300);
gdgrpcls();

/* Rideau River. */

gdgrpopn();
gdaset(514,64);
zzvec(30, 50);
zzvec(10, 60);
zzvec(-30, 110);
zzvec(-90, 180);
zzvec(-5, 50);
zzvec(10, 60);
zzvec(15, 40);
zzvec(40, 50);
zzvec(0, 20);
gdgrpcls();
```

AD-A153 634 A GENERAL AREA AIR TRAFFIC CONTROLLER SIMULATION USING COLOUR GRAPHICS(U) DEFENCE RESEARCH ESTABLISHMENT OTTAWA (ONTARIO) B J FORD SEP 83 DREO-890 2/2

UNCLASSIFIED F/G 17/7 NL





```
gdgrpopn();
gdaset(519,66);
zzvec(35, 45);
zzvec(5, 50);
zzvec(-10, 30);
zzvec(-15, 60);
zzvec(-20, 70);
zzvec(-80, 140);
zzvec(-10, 50);
zzvec(10, 55);
zzvec(30, 55);
zzvec(35, 30);
zzvec(7, 40);
gdgrpcls();

/* Gatineau River. */

gdgrpopn();
gdaset(330, 924);
zzvec(15, -25);
zzvec(25, -50);
zzvec(0, -45);
zzvec(10, -20);
zzvec(15, -5);
zzvec(10, -20);
zzvec(45, -25);
zzvec(35, -30);
zzvec(5, -15);
gdgrpcls();

gdgrpopn();
gdaset(333, 927);
zzvec(25, -50);
zzvec(20, -25);
zzvec(0, -40);
zzvec(10, -10);
zzvec(8, -10);
zzvec(20, -20);
```

```
zzvec(25, -20);
zzvec(50, -40);
zzvec(5, -15);
gdgrpcls();

/* Ottawa River. */

gdgrpopn();
gdaset(140, 759);
zzvec(80, -50);
zzvec(50, -60);
zzvec(15, -25);
zzvec(20, -10);
zzvec(60, 5);
zzvec(70, 30);
zzvec(50, 40);
zzvec(50, 30);
zzvec(20, 40);
zzvec(40, 20);
zzvec(70, 5);
zzvec(60, 40);
zzvec(45, 25);
zzvec(22, 20);
gdgrpcls();

gdgrpopn();
gdaset(120, 734);
zzvec(60, -45);
zzvec(40, -70);
zzvec(-10, -20);
zzvec(5, -25);
zzvec(15, -5);
zzvec(10, -10);
zzvec(-5, -10);
zzvec(10, -5);
zzvec(25, 10);
zzvec(10, -5);
zzvec(20, 0);
zzvec(25, 20);
```

```
zzvec(5, 20);
zzvec(10, 10);
zzvec(15, -5);
zzvec(35, 5);
zzvec(45, 30);
zzvec(30, 40);
zzvec(40, 25);
zzvec(60, 20);
zzvec(55, 35);
zzvec(75, 35);
zzvec(70, 20);
zzvec(40, 30);
zzvec(15, 10);
gdgrpcls();

/* Degrees. */

gdgrpopn();
gdaset(410, 951);
zzvec(8, -30);
zzivec(-7, 29);
zzvec(8, -30);
gdgrpcls();
gdgrpopn();
gdaset(410, 970);
gdtext("0", 1);
gdgrpcls();

gdgrpopn();
gdaset(660, 939);
zzvec(-8, -30);
zzivec(9, 31);
zzvec(-8, -30);
gdgrpcls();
gdgrpopn();
gdaset(660, 959);
gdtext("30", 2);
gdgrpcls();
```

```
gdgrpopn();
gdaset(847, 809);
zzvec(-17, -13);
zzivec(18, 14);
zzvec(-17, -13);
gdgrpcls();
gdgrpopn();
gdaset(867, 809);
gdtext("60", 2);
gdgrpcls();

gdgrpopn();
gdaset(948, 614);
zzvec(-30, -8);
zzivec(30, 9);
zzvec(-30, -8);
gdgrpcls();
gdgrpopn();
gdaset(968, 614);
gdtext("90", 2);
gdgrpcls();

gdgrpopn();
gdaset(945, 384);
zzvec(-30, 8);
zzivec(30, -7);
zzvec(-30, 8);
gdgrpcls();
gdgrpopn();
gdaset(965, 384);
gdtext("120", 3);
gdgrpcls();

gdgrpopn();
gdaset(820, 184);
zzvec(-17, 15);
zzivec(16, -14);
zzvec(-17, 15);
gdgrpcls();
```

```
gdgrpopn();
gdaset(840, 184);
gdtext("150", 3);
gdgrpcls();

gdgrpopn();
gdaset(610, 76);
zzvec(-5, 25);
zzivec(4, -25);
zzvec(-5, 25);
gdgrpcls();
gdgrpopn();
gdaset(610, 46);
gdtext("180", 3);
gdgrpcls();

gdgrpopn();
gdaset(365, 84);
zzvec(10, 30);
zzivec(-11, -31);
zzvec(10, 30);
gdgrpcls();
gdgrpopn();
gdaset(365, 54);
gdtext("210", 3);
gdgrpcls();

gdgrpopn();
gdaset(188, 204);
zzvec(25, 18);
zzivec(-26, -19);
zzvec(25, 18);
gdgrpcls();
gdgrpopn();
gdaset(138, 200);
gdtext("240", 3);
gdgrpcls();

gdgrpopn();
```

```
gdaset(75, 399);
zzvec(30, 5);
zzivec(-31, -6);
zzvec(30, 5);
gdgrpcls();
gdgrpopn();
gdaset(29, 399);
gdtext("270", 3);
gdgrpcls();

gdgrpopn();
gdaset(83, 639);
zzvec(30, -8);
zzivec(-29, 7);
zzvec(30, -8);
gdgrpcls();
gdgrpopn();
gdaset(33, 643);
gdtext("300", 3);
gdgrpcls();

gdgrpopn();
gdaset(195, 834);
zzvec(25, -22);
zzivec(-24, 21);
zzvec(25, -22);
gdgrpcls();
gdgrpopn();
gdaset(165, 834);
gdtext("330", 3);
gdgrpcls();

/* Weather. */

gdgrpopn();          /* Large mass outline. */
gdaset(750, 724);
zzvec(80, -20);
zzvec(20, -40);
zzvec(0, -50);
```

```
zzvec(-10, -40);
zzvec(-20, -30);
zzvec(-30, 0);
zzvec(-20, 30);
zzvec(-20, 10);
zzvec(-30, 60);
zzvec(10, 60);
zzvec(20, 20);
gdgrpcls();

gdgrpopn();          /* large mass lines. */
gdaset(740, 714);
zzvec(100, -25);
zzivec(-111, -6);
zzvec(120, -30);
zzivec(-127, -6);
zzvec(128, -32);
zzivec(-120, -6);
zzvec(112, -28);
zzivec(-92, -6);
zzvec(80, -20);
gdgrpcls();

gdgrpopn();          /* Small mass outline. */
gdaset(750, 574);
zzvec(30, -10);
zzvec(0, -30);
zzvec(-25, -20);
zzvec(-10, 10);
zzvec(-20, 30);
zzvec(25, 20);
gdgrpcls();

gdgrpopn();
gdaset(740, 564);
zzvec(40, -10);
zzivec(-43, -13);
zzvec(38, -8);
gdgrpcls();
```

```
gdgrpopn();          /* Arrow. */
gdaset(733, 544);
zzvec(-40, -20);
zzivec(39, 19);
zzvec(-40, -20);
zzivec(13, 13);
zzvec(-15, -15);
zzivec(15, 14);
zzvec(-15, -15);
zzivec(21, 3);
zzvec(-20, -1);
zzivec(20, 0);
zzvec(-20, -1);
gdgrpcls();

/* Mountains. */

gdgrpopn();
gdaset(360, 744);
zzvec(20, 0);
zzvec(10, -20);
zzvec(0, -20);
zzvec(-20, 0);
zzvec(-10, 40);
gdgrpcls();

gdgrpopn();
gdaset(350, 774);
zzvec(25, 0);
zzvec(15, -30);
zzvec(10, -30);
zzvec(-5, -40);
zzvec(-5, 0);
zzvec(-20, 10);
zzvec(-10, 10);
zzvec(-20, 30);
zzvec(10, 50);
gdgrpcls();
```

```
gdgrpopn();
gdaset(300, 824);
zzvec(35, -10);
zzvec(30, -10);
zzvec(20, -30);
zzvec(15, -40);
zzvec(10, -40);
zzvec(0, -35);
zzvec(-40, 0);
zzvec(-20, 30);
zzvec(-25, 35);
zzvec(-10, 40);
zzvec(-5, 15);
zzvec(-15, 20);
zzvec(-20, 25);
zzvec(25, 0);
gdgrpcls();

gdgrpopn();
gdaset(385, 744);
gdtext("1150", 4);
gdgrpcls();

/* Beacons. */

/* VORTAC */

tempx = 215;
tempy = 744;
for(count = 0; count < 10; count++)          /* 10 groups. */
{    gdgrpopn();
     gdaset(tempx, tempy);
     tempx++;
     tempy++;
     zzvec(10, -10);
     gdgrpcls();
}
gdgrpopn();
```

```
zzivec(10, -10);
zzvec(18, 0);
gdgrpcls();
tempx =+ 38;
for(count = 0; count < 10; count++)          /* 10 more groups. */
{    gdgrpopn();
     gdaset(tempx, tempy);
     tempx++;
     tempy--;
     zzvec(-10, -10);
     gdgrpcls();
}
gdgrpopn();
zzivec(-10, -10);
zzvec(-10, -10);
gdgrpcls();
tempx =- 20;
tempy =- 20;
for(count = 0; count < 15; count =+ 2)          /* 8 more groups. */
{    gdgrpopn();
     gdaset(tempx, tempy);
     tempx =- 2;
     zzvec(0, -15);
     gdgrpcls();
}
gdgrpopn();
gdaset(tempx, tempy);
zzvec(-10, 10);
gdgrpcls();
gdgrpopn();
gdaset(245, 734);
zzvec(0, -1);
zzvec(0, -1);
zzvec(0, -1);
gdgrpcls();

/* TACAN */

gdgrpopn();
```

```
gdaset(440, 484);
zzvec(7, 7);
zzvec(10, -10);
zzvec(20, 0);
zzvec(10, 10);
zzvec(7, -7);
zzvec(-10, -10);
zzvec(-7, -15);
zzvec(0, -10);
zzvec(-20, 0);
zzvec(0, 10);
zzvec(-7, 15);
zzvec(-10, 10);
gdgrpcls();
gdgrpopn();
gdaset(467, 469);
zzvec(0, -1);
zzvec(0, -1);
zzvec(0, -1);
gdgrpcls();

/* ndb */

gdgrpopn();
gdaset(724, 349);
zzvec(1, 0);
zzvec(1, 0);
zzvec(1, 0);
gdgrpcls();
gdgrpopn();
gdcirc(725, 349, 5, 16, 32);
gdgrpcls();
gdgrpopn();
gdcirc(725, 349, 6, 16, 32);
gdgrpcls();
gdgrpopn();
gdcirc(725, 349, 7, 16, 32);
gdgrpcls();
gdgrpopn();
```

```
gdcirc(725, 349, 20, 32, 64);
gdgrpcls();

/* ndb */

gdgrpopn();
gdaset(679, 599);
zzvec(1, 0);
zzvec(1, 0);
zzvec(1, 0);
gdgrpcls();
gdgrpopn();
gdcirc(680, 599, 5, 16, 32);
gdgrpcls();
gdgrpopn();
gdcirc(680, 599, 6, 16, 32);
gdgrpcls();
gdgrpopn();
gdcirc(680, 599, 7, 16, 32);
gdgrpcls();
gdgrpopn();
gdcirc(680, 599, 20, 32, 64);
gdgrpcls();

/* Airways. */

gdgrpopn();
gddash(330., 99., 685., 584.);
gdgrpcls();
gdgrpopn();
gddash(90., 659., 670., 604.);
gdgrpcls();
gdgrpopn();
gddash(550., 959., 685., 619.);
gdgrpcls();
gdgrpopn();
gddash(945., 624., 700., 604.);
gdgrpcls();
```

```
/* Runways. */

gdgrpopn();
gdcolr(3);
gdaset(540, 484);
zzvec(-75, 55);
zzivec(74, -56);
zzvec(-75, 55);

zzivec(56, -59);
zzvec(60, 40);
zzivec(-60, -41);
zzvec(60, 40);

zzivec(-110, 0);
zzvec(15, 50);
zzivec(-14, -50);
zzvec(15, 50);

zzivec(-10, -30);
zzvec(30, 30);
zzivec(-30, -31);
zzvec(30, 30);
gdgrpcls();

/* Tower. */

gdgrpopn();
gdaset(500, 519);
gdtext("T", 1);
gdgrpcls();

/* Aircraft. */

gdgrpopn();
gdcolr(1);
plane(560,464);
gdgrpcls();
gdgrpopn();
```

```
gdaset(570, 490);
gdtext("AC 111", 6);
gdgrpcls();
gdgrpopn();
gdaset(570, 474);
gdtext("80", 2);
gdgrpcls();

gdgrpopn();
plane(620, 524);
gdgrpcls();
gdgrpopn();
gdaset(630, 550);
gdtext("GX 470", 6);
gdgrpcls();
gdgrpopn();
gdaset(630, 534);
gdtext("85", 2);
gdgrpcls();

gdgrpopn();
plane(310, 254);
gdgrpcls();
gdgrpopn();
gdaset(320, 280);
gdtext("EA 123", 6);
gdgrpcls();
gdgrpopn();
gdaset(320, 264);
gdtext("70", 2);
gdgrpcls();

gdgrpopn();
plane(260, 279);
gdgrpcls();
gdgrpopn();
gdaset(270, 305);
gdtext("AC 441", 6);
gdgrpcls();
```

```
gdgrpopn();
gdaset(270, 289);
gdtext("60", 2);
gdgrpcls();

gdgrpopn();
gdcolr(2);
plane(800, 224);
gdgrpcls();
gdgrpopn();
gdaset(810, 250);
gdtext("GX 741", 6);
gdgrpcls();
gdgrpopn();
gdaset(810, 234);
gdtext("90", 2);
gdgrpcls();

gdgrpopn();
plane(700, 324);
gdgrpcls();
gdgrpopn();
gdaset(710, 350);
gdtext("CP 899", 6);
gdgrpcls();
gdgrpopn();
gdaset(710, 334);
gdtext("75", 2);
gdgrpcls();

gdgrpopn();
gdcolr(0);
plane(450, 864);
gdgrpcls();
gdgrpopn();
gdaset(460, 890);
gdtext("AC 164", 6);
gdgrpcls();
gdgrpopn();
```

```
gdaset(460, 874);
gdtext("70", 2);
gdgrpcls();

gdgrpopn();
plane(480, 864);
gdgrpcls();
gdgrpopn();
gdaset(490, 890);
gdtext("AC 159", 6);
gdgrpcls();
gdgrpopn();
gdaset(490, 874);
gdtext("70", 2);
gdgrpcls();
gdgrpend();


for (count = 0; count < 119; count++)
     garray[count] = count + 1;
gdupdate(garray, 119);   /* Display all of the above information. */

/* The x coordinates of the initial plane positions. */

xc[1] = 560;
xc[2] = 620;
xc[3] = 310;
xc[4] = 260;
xc[5] = 800;
xc[6] = 700;
xc[7] = 450;
xc[8] = 480;

/* The y coordinates of the initial plane positions. */

yc[1] = 464;
yc[2] = 524;
yc[3] = 254;
yc[4] = 279;
```

```
yc[5] = 224;
yc[6] = 324;
yc[7] = 864;
yc[8] = 864;

/* The initial aircraft speeds.  Speeds are translated to "p" values
     which are actually the number of rasters skipped in each move. */

p[1] = 10;
p[2] = 15;
p[3] = 18;
p[4] = 12;
p[5] = 20;
p[6] = 18;
p[7] = 5;
p[8] = 5;

/* The following are the slopes of the lines along which the aircraft
     travel.  These slopes may be changed by the atc.                  */

m[1] = .1;
m[2] = -10.;
m[3] = 1.;
m[4] = 20.;
m[5] = -1.;
m[6] = 30.;
m[7] = 10.;
m[8] = -10.;

/* These values are used in the quadratic equation that determines the
     aircraft's next position, to indicate whether the aircraft is to
     head in a positive direction or a negative direction.             */

plormin[1] = 1;
plormin[2] = 1;
plormin[3] = 1;
plormin[4] = 1;
plormin[5] = 1;
plormin[6] = 1;
```

```
plormin[7] = -1;
plormin[8] = -1;

printf("\nAny aircraft may have its speed and/or bearing changed.\n");
printf("The format is as follows:\n");
printf("A B C\n");
printf("where A is the flight number of the aircraft (eg. AC_123),\n");
printf("      B is the speed in miles per hour of the aircraft\n");
printf("        (positive integer),\n");
printf("      C is the bearing of the aircraft\n");
printf("        (integer value between 0 and 359),\n");
printf(": ");

for (move = 1; move <= 40; move++)      /* The aircraft move 40 times. */
{
     /* New aircraft positions. */

     for (plncnt = 1; plncnt <= 8; plncnt++)      /* For each of 8 aircraft. */

          /* The following set of equations calculate the next position
               of the aircraft knowing the slope (m), the number of
               rasters to skip (p), the direction (plormin), and the
               current x and y coordinates (xc and yc).  The quadratic
               equation is used.                                        */

     {    a = 1 + 1 / ( m[plncnt] * m[plncnt] );

          b = -2 * yc[plncnt] * a;

          c = yc[plncnt] * (yc[plncnt] * a) - p[plncnt] * p[plncnt];

          ynfloat = (-b + plormin[plncnt] * pow (( b * b - 4 * a * c ),
                    .5 )) / ( 2 * a );

          /* The value must be converted to integer to refer to a
               screen position.                                        */

          ynfloat = ynfloat + .5;
          yn[plncnt] = ynfloat;
```

```
        /* The x value is obtained knowing the slope is the change
             in the y value divided by the change in the x value.   */

        xnfloat = ( yn[plncnt] + m[plncnt] * xc[plncnt] - yc[plncnt] )
                        / m[plncnt];

        /* The value must be converted to integer to refer to a
             screen position.                                       */

        xnfloat = xnfloat + .5;
        xn[plncnt] = xnfloat;

}

/* Check which aircraft are within miss distance (red) and which
     are within twice miss distance (orange).  All aircraft
     initially set to standard amber.                              */

for (plncnt = 1; plncnt <= 8; plncnt++)
     colour[plncnt] = 2;

/* All possible pairs of aircraft are considered. */

for (plncnt = 1; plncnt <= 8; plncnt++)
{    for (inner = plncnt + 1; inner <= 8; inner ++)
     {
          x1 = xn[plncnt];
          x2 = xn[inner];
          y1 = yn[plncnt];
          y2 = yn[inner];

          /* The distance between the two aircraft is determined. */

          dist = pow ((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2), .5);
          if (dist <= 40.)    /* Approximately miss distance. */
          {    colour[plncnt] = 0;
               colour[inner] = 0;
          }
```

```
            else if (dist <= 80.)    /* Twice miss distance. */
                 {    if (colour[plncnt] > 1)
                           colour[plncnt] = 1;
                      if ( colour[inner] > 1)
                           colour[inner] = 1;
                 }
       }
}

/* Edit plane and information position, and colour. */

group = 96;    /* The first group that needs editing. */

for (plncnt = 1; plncnt <= 8; plncnt++)

       /* First check that the aircraft is still on the screen. */

{      if ((xn[plncnt] <= 974) & (xn[plncnt] >= 10) &
            (yn[plncnt] <= 974) & (yn[plncnt] >= 10))

            /* The plane gets the new position. */

       {    gdedaset(group, 1, xn[plncnt], yn[plncnt]);

            /* The plane gets the current colour. */

            gdedcolr(group, 1, colour[plncnt]);

            /* The information gets its new position and colour. */

            gdedaset(group + 1, 1, xn[plncnt] + 10, yn[plncnt] + 26);
            gdedcolr(group + 1, 1, colour[plncnt]);
            gdedaset(group + 2, 1, xn[plncnt] + 10, yn[plncnt] + 10);
            gdedcolr(group + 2, 1, colour[plncnt]);

            /* The new values become the current values. */

            xc[plncnt] = xn[plncnt];
            yc[plncnt] = yn[plncnt];
```

```
        }
        else

             /* If the aircraft is out of bounds then no longer
                  display it.                                        */

        {    gdedints(group, 1, 0);
             gdedints(group + 1, 1, 0);
             gdedints(group + 2, 1, 0);
        }
        group =+ 3;
}

/* Check to see if there has been any atc interaction. */

if (flag == 1)
{    flag = 0;
     if( (nbytes = read(0, buf, 70)) <= 1)   /* Valid interaction? */
          continue;

     /* Read the interactive input into the appropriate variables. */

     scanf(-1, buf, "%s %d %d", &fltno, &speed, &bearing);

     /* Determine the number of the plane from its flight number. */

     if (compare(fltno, "AC_111") == 1) subscr = 1;
     else if (compare(fltno, "GX_470") == 1) subscr = 2;
          else if (compare(fltno, "EA_123") == 1) subscr = 3;
               else if (compare(fltno, "AC_441") == 1) subscr = 4;
                    else if (compare(fltno, "GX_741") == 1)
                              subscr = 5;
                         else if (compare(fltno, "CP_899") == 1)
                                   subscr = 6;
                              else if (compare(fltno, "AC_164") == 1)
                                        subscr = 7;
                                   else if (compare(fltno, "AC_159")
                                             == 1) subscr = 8;
                                        else
```

```
                                                  {
printf("There is an error in the flight number,  please try again.\n");
printf(": ");
                                                       subscr = 0;
                                                  }
               if (subscr != 0)    /* If a valid flight number. */

                    /* Determine the degree from the bearing.  The degree
                         will be used to determine the new slope of the
                         line along which the aircraft travels.           */

               {    if ((13 <= bearing) & (bearing <= 103))
                         degree = 103 - bearing;
                    else if (((104 <= bearing) & (bearing <= 359)) ^
                              ((0 <= bearing) & ( bearing <= 12)))
                              degree = 463 - bearing;
                         else
                         {
printf("There is an error in the bearing, please try again.\n");
printf(": ");
                              bearing = -1;
                         }
                    if (bearing != -1)  /* If a valid bearing. */

                         /* sin and cos are calculated using radians so
                              degrees are changed to radians.          */

                    {    radian = degree * .01745;

                         /* The new slope of the line along which the
                              aircraft travels.                        */

                         m[subscr] = sin(radian) / cos(radian);

                         /* The following determines if the aircraft travels
                              in a positive or a negative direction.       */

                         if (((0 <= bearing) & (bearing <= 103)) ^
                              ((283 <= bearing) & (bearing <= 359)))
```

```
                    plormin[subscr] = 1;
               else plormin[subscr] = -1;

               /* Speed is changed to a floating point value then
                    speed in terms of rasters to skip is calculated. */

               spf = speed;
               spf = spf / 30;
               p[subscr] = spf;
          }
     }
  }
}
```

# APPENDIX 3

## atcsec.c:
## THE ROUTINE THAT DISPLAYS INFORMATION OF SECONDARY IMPORTANCE TO THE AIR TRAFFIC CONTROLLER

```
/* The following draws the secondary display for the general area
     air traffic controller simulation.                          */


extern gdunit;
main(argc, argv)
int argc;
char *argv[];
{    int count, garray[100];

     if (argc > 1) gdunit = 1;          /* Set for VGS. */
     gdinit();
     gdsubend();

     /* Set up the titles. */

     gdgrpopn();
     gdcset(1);
     gdaset(20, 1000);
     gdtext("id", 2);
     gdgrpcls();
     gdgrpopn();
     gdaset(20, 995);
     gdtext("__", 2);
     gdgrpcls();
     gdgrpopn();
     gdaset(120, 1000);
     gdtext("Flight Level", 12);
     gdgrpcls();
     gdgrpopn();
     gdaset(120, 995);
     gdtext("____________", 12);
     gdgrpcls();
     gdgrpopn();
     gdaset(300, 1000);
     gdtext("Cleared FL", 10);
     gdgrpcls();
     gdgrpopn();
     gdaset(300, 995);
```

```
gdtext("__________", 10);
gdgrpcls();
gdgrpopn();
gdaset(450, 1000);
gdtext("Type", 4);
gdgrpcls();
gdgrpopn();
gdaset(450, 995);
gdtext("____", 4);
gdgrpcls();
gdgrpopn();
gdaset(550, 1000);
gdtext("Source", 6);
gdgrpcls();
gdgrpopn();
gdaset(550, 995);
gdtext("______", 6);
gdgrpcls();
gdgrpopn();
gdaset(670, 1000);
gdtext("Destination", 11);
gdgrpcls();
gdgrpopn();
gdaset(670, 995);
gdtext("___________", 11);
gdgrpcls();
gdgrpopn();
gdaset(850, 1000);
gdtext("TOA", 3);
gdgrpcls();
gdgrpopn();
gdaset(850, 995);
gdtext("___", 3);
gdgrpcls();
gdgrpopn();
gdaset(950, 1000);
gdtext("Speed", 5);
gdgrpcls();
gdgrpopn();
```

```
gdaset(950, 995);
gdtext("_____", 5);
gdgrpcls();

/* Set up the flight number (id). */

gdgrpopn();
gdcolr(4);
gdaset(10, 900);
gdtext("AC 111", 6);
gdgrpcls();
gdgrpopn();
gdaset(10, 800);
gdtext("GX 470", 6);
gdgrpcls();
gdgrpopn();
gdaset(10, 700);
gdtext("EA 123", 6);
gdgrpcls();
gdgrpopn();
gdaset(10, 600);
gdtext("AC 441", 6);
gdgrpcls();
gdgrpopn();
gdaset(10, 500);
gdtext("GX 741", 6);
gdgrpcls();
gdgrpopn();
gdaset(10, 400);
gdtext("CP 899", 6);
gdgrpcls();
gdgrpopn();
gdaset(10, 300);
gdtext("AC 164", 6);
gdgrpcls();
gdgrpopn();
gdaset(10, 200);
gdtext("AC 159", 6);
gdgrpcls();
```

```
/* Set up the flight levels. */

gdgrpopn();
gdcolr(3);
gdaset(140, 900);
gdtext("80", 2);
gdgrpcls();
gdgrpopn();
gdaset(140, 800);
gdtext("85", 2);
gdgrpcls();
gdgrpopn();
gdaset(140, 700);
gdtext("70", 2);
gdgrpcls();
gdgrpopn();
gdaset(140, 600);
gdtext("60", 2);
gdgrpcls();
gdgrpopn();
gdaset(140, 500);
gdtext("90", 2);
gdgrpcls();
gdgrpopn();
gdaset(140, 400);
gdtext("75", 2);
gdgrpcls();
gdgrpopn();
gdaset(140, 300);
gdtext("70", 2);
gdgrpcls();
gdgrpopn();
gdaset(140, 200);
gdtext("70", 2);
gdgrpcls();

/* Set the cleared flight level. */
```

```
gdgrpopn();
gdcolr(4);
gdaset(320, 600);
gdtext("70", 2);
gdgrpcls();

/* Set up the type of aircraft information. */

gdgrpopn();
gdcolr(3);
gdaset(450, 900);
gdtext("DC9", 3);
gdgrpcls();
gdgrpopn();
gdaset(450, 800);
gdtext("B707", 4);
gdgrpcls();
gdgrpopn();
gdaset(450, 700);
gdtext("DC10", 4);
gdgrpcls();
gdgrpopn();
gdaset(450, 600);
gdtext("B737", 4);
gdgrpcls();
gdgrpopn();
gdaset(450, 500);
gdtext("DASH7", 5);
gdgrpcls();
gdgrpopn();
gdaset(450, 400);
gdtext("DC8", 3);
gdgrpcls();
gdgrpopn();
gdaset(450, 300);
gdtext("B707", 4);
gdgrpcls();
gdgrpopn();
gdaset(450, 200);
```

```
gdtext("DC9", 3);
gdgrpcls();

/* Set up the source information of the aircraft's route. */

gdgrpopn();
gdcolr(4);
gdaset(560, 900);
gdtext("YOW", 3);
gdgrpcls();
gdgrpopn();
gdaset(560, 800);
gdtext("YOW", 3);
gdgrpcls();
gdgrpopn();
gdaset(560, 700);
gdtext("YYZ", 3);
gdgrpcls();
gdgrpopn();
gdaset(560, 600);
gdtext("YQR", 3);
gdgrpcls();
gdgrpopn();
gdaset(560, 500);
gdtext("YOW", 3);
gdgrpcls();
gdgrpopn();
gdaset(560, 400);
gdtext("YYZ", 3);
gdgrpcls();
gdgrpopn();
gdaset(560, 300);
gdtext("YWG", 3);
gdgrpcls();
gdgrpopn();
gdaset(560, 200);
gdtext("YEG", 3);
gdgrpcls();
```

```
/* Set up the destination information of the aircraft's route. */

gdgrpopn();
gdcolr(3);
gdaset(700, 900);
gdtext("YMX", 3);
gdgrpcls();
gdgrpopn();
gdaset(700, 800);
gdtext("YYZ", 3);
gdgrpcls();
gdgrpopn();
gdaset(700, 700);
gdtext("BWI", 3);
gdgrpcls();
gdgrpopn();
gdaset(700, 600);
gdtext("YMX", 3);
gdgrpcls();
gdgrpopn();
gdaset(700, 500);
gdtext("YXU", 3);
gdgrpcls();
gdgrpopn();
gdaset(700, 400);
gdtext("YMX", 3);
gdgrpcls();
gdgrpopn();
gdaset(700, 300);
gdtext("YOW", 3);
gdgrpcls();
gdgrpopn();
gdaset(700, 200);
gdtext("YOW", 3);
gdgrpcls();

/* Set up the time of arrival at destination information. */

gdgrpopn();
```

```
gdcolr(4);
gdaset(840, 900);
gdtext("14:17", 5);
gdgrpcls();
gdgrpopn();
gdaset(840, 800);
gdtext("14:15", 5);
gdgrpcls();
gdgrpopn();
gdaset(840, 700);
gdtext("16:08", 5);
gdgrpcls();
gdgrpopn();
gdaset(840, 600);
gdtext("14:30", 5);
gdgrpcls();
gdgrpopn();
gdaset(840, 500);
gdtext("15:31", 5);
gdgrpcls();
gdgrpopn();
gdaset(840, 400);
gdtext("14:10", 5);
gdgrpcls();
gdgrpopn();
gdaset(840, 300);
gdtext("13:24", 5);
gdgrpcls();
gdgrpopn();
gdaset(840, 200);
gdtext("13:12", 5);
gdgrpcls();

/* Set up the speed of the aircraft. */

gdgrpopn();
gdcolr(3);
gdaset(950, 900);
gdtext("300", 3);
```

```
        gdgrpcls();
        gdgrpopn();
        gdaset(950, 800);
        gdtext("450", 3);
        gdgrpcls();
        gdgrpopn();
        gdaset(950, 700);
        gdtext("540", 3);
        gdgrpcls();
        gdgrpopn();
        gdaset(950, 600);
        gdtext("360", 3);
        gdgrpcls();
        gdgrpopn();
        gdaset(950, 500);
        gdtext("600", 3);
        gdgrpcls();
        gdgrpopn();
        gdaset(950, 400);
        gdtext("540", 3);
        gdgrpcls();
        gdgrpopn();
        gdaset(950, 300);
        gdtext("150", 3);
        gdgrpcls();
        gdgrpopn();
        gdaset(950, 200);
        gdtext("150", 3);
        gdgrpcls();
        gdgrpend();

        for(count = 0; count <= 72; count++)
             garray[count] = count + 1;
        gdupdate(garray, 73);
}
```

# APPENDIX 4

# AIRPORT CODES

BWI - Baltimore

LGX - London, England

MIA - Miami

YEG - Edmonton

YMX - Mirabel

YOW - Ottawa

YQB - Quebec City

YQG - Windsor

YQR - Regina

YTZ - Toronto Island

YVR - Vancouver

YWG - Winnipeg

YXU - London, Ontario

YYC - Calgary

YYG - Charlottetown

YYZ - Toronto

# APPENDIX 5

## gddash.c:
## THE ROUTINE TO DRAW DASHED LINES

```
gddash(xstart, ystart, xend, yend)

/* This routine draws a dashed line from the point indicated by
     (xstart, ystart) to the point indicated by (xend, yend).    */

float xstart,        /* The start x value. */
     ystart,         /* The start y value. */
     xend,           /* The last x value. */
     yend;           /* The last y value. */
{    int  dx,        /* The change in x to keep a constant size dash. */
          dy,        /* The change in y to keep a constant size dash. */
          delta,     /* A positive or negative change in x or y. */
          xtemp,     /* x temporarily stored to calculate relative x value. */
          ytemp,     /* y temporarily stored to calculate relative y value. */
          x,         /* The x value that is incremented in the loop. */
          y,         /* The y value that is incremented in the loop. */
          xyrel[2]; /* Relative xy array used to draw a vector. */

     float b,        /* The y intercept. */
           m;        /* The slope. */

     extern float cos(), sin(), atan();

     /* Take care of the case of an infinite slope. */

     if (xend == xstart) m = 29999;
     else  m = (yend - ystart) / (xend - xstart);
     if (xend == xstart) b = 29999;
     else  b = (xend * ystart - xstart * yend) / (xend - xstart);

     if ((m < 1) && (m > -1))       /* Is the line of a horizontal nature? */
     {    dx = 25 * cos(atan(m));  /* The length of the dash can be modified
                                       by changing 25 to the desired number
                                       of rasters in the dash.            */

          x = xstart;
          y = ystart;
          if (xstart < xend)             /* Left to Right. */
          {    delta = dx;
               gdaset(x,y);
```

```
            while (x <= (xend - delta))
            {     xtemp = x;
                  ytemp = y;
                  x =+ delta;
                  y = b + m * x;       /* New absolute x and y values. */
                  xyrel[0] = x - xtemp;
                  xyrel[1] = y - ytemp;
                  gdvect(xyrel, 1);    /* Draw dash. */
                  xtemp = x;
                  ytemp = y;
                  x =+ delta;
                  y = b + m * x;       /* Skip dash. */
                  xyrel[0] = x - xtemp;
                  xyrel[1] = y - ytemp;
                  gdivec(xyrel, 1);
            }
      }
      else                             /* Right to Left. */
      {     delta = -dx;
            gdaset(x, y);
            while (x >= (xend - delta))
            {     xtemp = x;
                  ytemp = y;
                  x =+ delta;
                  y = b + m * x;       /* New absolute x and y values. */
                  xyrel[0] = x - xtemp;
                  xyrel[1] = y - ytemp;
                  gdvect(xyrel, 1);    /* Draw dash. */
                  xtemp = x;
                  ytemp = y;
                  x =+ delta;
                  y = b + m * x;       /* Skip dash. */
                  xyrel[0] = x - xtemp;
                  xyrel[1] = y - ytemp;
                  gdivec(xyrel, 1);
            }
      }
}
else      /* The line is of a vertical nature. */
```

```
{     dy = 25 * sin(atan(m));
      if (dy < 0) dy = -dy;
      x = xstart;
      y = ystart;
      if (ystart < yend)              /* Up to Down. */
      {     delta = dy;
            gdaset(x, y);
            while (y <= (yend - delta))
            {     xtemp = x;
                  ytemp = y;
                  y =+ delta;
                  if (m == 29999) x = xtemp;
                  else x = (y - b) / m;     /* New absolute x and y values. */
                  xyrel[0] = x - xtemp;
                  xyrel[1] = y - ytemp;
                  gdvect(xyrel, 1);   /* Draw dash. */
                  xtemp = x;
                  ytemp = y;
                  y =+ delta;
                  if (m == 29999) x = xtemp;
                  else x = (y - b) / m;     /* Skip dash. */
                  xyrel[0] = x - xtemp;
                  xyrel[1] = y - ytemp;
                  gdivec(xyrel, 1);
            }
      }
      else                            /* Down to Up. */
      {     delta = -dy;
            gdaset(x, y);
            while (y >= (yend - delta))
            {     xtemp = x;
                  ytemp = y;
                  y =+ delta;
                  if (m == 29999) x = xtemp;
                  else x = (y - b) / m;     /* New absolute x and y values. */
                  xyrel[0] = x - xtemp;
                  xyrel[1] = y - ytemp;
                  gdvect(xyrel, 1);   /* Draw dash. */
                  xtemp = x;
```

```
                        ytemp = y;
                        y =+ delta;
                        if (m == 29999) x = xtemp;
                        else x = (y - b) / m;     /* Skip dash. */
                        xyrel[0] = x - xtemp;
                        xyrel[1] = y - ytemp;
                        gdivec(xyrel, 1);
                    }
                }
            }
        }
```

# APPENDIX 6

## gdcirc.c:
## THE ROUTINE TO DRAW CIRCLES

```
/*
 * CIRCLE - draw a circle with center at VVX,VVY with radius VVRAD.
 *        - use VVINC vectors to draw the circle.
 *        - skip a vector every VVSKP vector
 *             (i.e. if a dashed circle is desired).
 */

gdcirc(vvx, vvy, vvrad, vvinc, vvskp)
int  vvx;            /* X co-ordinate of center of circle.   */
int  vvy;            /* Y co-ordinate of center of circle.   */
int  vvrad;          /* radius of the circle.                */
int  vvinc;          /* number of vectors used for circle.   */
int  vvskp;          /* skip a vector every VVSKP vectors.   */
     {
     extern double sin();     /* externally define sine function.              */
     extern double cos();     /* externally define cosine function.            */

     double vvfx;             /* X value returned from the cosine function.    */
     double vvfy;             /* Y value returned from the sine function.      */

     double vvtheta;          /* angle used for calculations.                  */

     double vvtwopie;         /* initialized to two times pi.                  */
     double vvthetinc;        /* angle increment.                              */
     double vvfrad;           /* floating point radius.                        */

     int  vvx1,vvy1;          /* old X and Y values.                           */
     int  vvx2,vvy2;          /* new X and Y values.                           */
     int  vvdx,vvdy;          /* X and Y displacement values.                  */

     int varray[2];           /* Used for drawing vectors.                     */
     int  vvcnt;              /* skip counter.                                 */


     vvtwopie = 6.283185307;           /* Twice pi. */
     vvthetinc = vvtwopie / vvinc;     /* initialize angle increment.          */
     vvfrad = vvrad;                   /* convert radius to floating point.    */

     gdaset(vvx + vvrad, vvy);         /* set start point to right side of circle.  */
```

```
vvx1 = vvrad;                          /* initialize starting X to the radius.      */
vvy1 = 0;                              /* initialize starting Y to 0.               */

vvcnt = 0;                             /* intialize skip counter to 0.              */


                                       /* loop calculating and writing vectors.     */
for(vvtheta = vvthetinc; vvtheta <= vvtwopie; vvtheta =+ vvthetinc)
     {
     vvfx = cos(vvtheta) * vvfrad; /* get X value.                                  */
     vvfy = -sin(vvtheta) * vvfrad;/* get Y value.                                  */

     vvx2 = vvfx;                      /* convert floating value to integer.        */
     vvy2 = vvfy;                      /* convert floating value to integer.        */

     vvdx = vvx2 - vvx1;               /* calculate X displacement.                 */
     vvdy = vvy2 - vvy1;               /* calculate Y displacement.                 */


     vvx1 = vvx2;                      /* set old X, to new X.                      */
     vvy1 = vvy2;                      /* set old Y, to new Y.                      */

     varray[0] = vvdx;
     varray[1] = vvdy;

     if(++vvcnt > vvskp)               /* check if it is time to skip a vector.     */
          {
          vvcnt = 0;                   /* reset counter to 0.                       */
                                       /* reposition starting point of next vector. */
          gdivec(varray, 1);
          }
     else
          {
          gdvect(varray, 1);           /* draw the next vector.                     */
          }
     }
}
```

# APPENDIX 7

## inread.c:
## THE ROUTINE TO HANDLE INTERACTIVE INPUT

```
/* This program can be used to provide an asyncronous input capability
 * for terminal input for use with a user program.  It achieves this by
 * running two processes connected via a pipe.  This program runs in an
 * infinite loop reading terminal input and writing it via a pipe to the
 * standard input of the user process.  Whenever input is available to
 * the user program it is notified via a signal 16.  The user program
 * must be set up to catch the signal.  Normally the user process would
 * set a flag on the receipt of the signal.  The user's program would
 * intermittently check the flag to determine when to read.
 *
 * The user's program is invoked via an "inread user-prog arg1,...,argn".
 * The user program will be executed with the given arguments.
 */

int  pfid[2];        /* 2 word array to contain pipe fid's. */
int  forkid;         /* fork id in parent process, 0 in child. */

int  ifid;           /* standard input file id (i.e. terminal). */
int  ofid;           /* set to pipe fid for write to other process. */

char buf[80];        /* I/O buffer. */
int  nbytes;         /* number of bytes read or written. */

int  nargc;          /* counter for argument copy. */
char *nargv[10];     /* array to copy arguments into. */

main(argc, argv)
int  argc;
char *argv[];
     {

     if(argc < 2)        /* make sure there are some arguments. */
          {
          printf("%s : arg count\n", argv[0]);
          exit(-1);
          }

     /* copy arguments into argument array in preparation for the execute. */
```

```
for(nargc = 0; nargc < argc; nargc++)
     {
     nargv[nargc] = argv[nargc+1];
     }
nargv[argc-1] = 0;  /* properly terminate the argument array. */

ifid = dup(0);      /* make a copy of the standard input fid. */
close(0);           /* close the original standard input. */
pipe(pfid);         /* open a pipe, the read fid will be 0. */
ofid = pfid[1];     /* get pipe write fid. */

/* do a fork and exit if an error. */

if( (forkid = fork()) < 0)
     {
     printf("%s : can not fork\n", argv[0]);
     exit(-1);
     }

/* if we are in the child process execute the program given as the
 * first argument to this program.
 */
if(forkid == 0)
     {
     execv(nargv[0], &nargv[0]);
     printf("%s : can not execv : %s\n", argv[0], nargv[0]);
     exit(-1);
     }

/* if we get here we must be in the parent process. */
/* all set up, so loop away.  */
while(1)
     {
     printf(": ");  /* prompt the operator for input. */

     /* read an input line from the terminal. */

     if( (nbytes = read(ifid, buf, 80)) < 0)
          {
```

```
            printf("%s : read error\n", argv[0]);
            exit(-1);
            }

        /* exit on an EOF. */

        if(nbytes == 0) exit(0);

        /* write the line from the terminal to the other process. */

        if(write(ofid, buf, nbytes) < 0)
            {
            printf("%s: write error\n", argv[0]);
            exit(-1);
            }

        /* notify the other process that it has input by sending a
         * signal 16.  The other process must be set up to catch
         * this signal, or it will be killed.
         */

        if(kill(forkid, 16) < 0)
            {
            printf("%s : kill error : %d\n", argv[0], forkid);
            }
        }
}
```

# APPENDIX 8

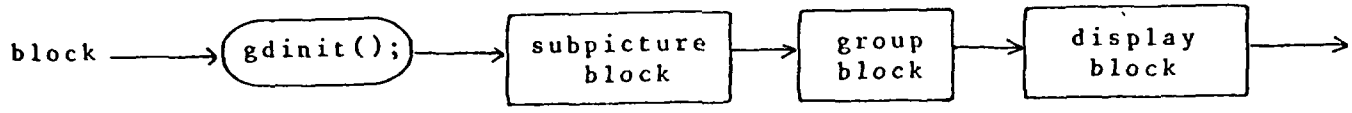
## ininit.c:
## THE ROUTINE TO RECEIVE THE SIGNAL
## THAT INTERACTIVE INPUT IS AVAILABLE

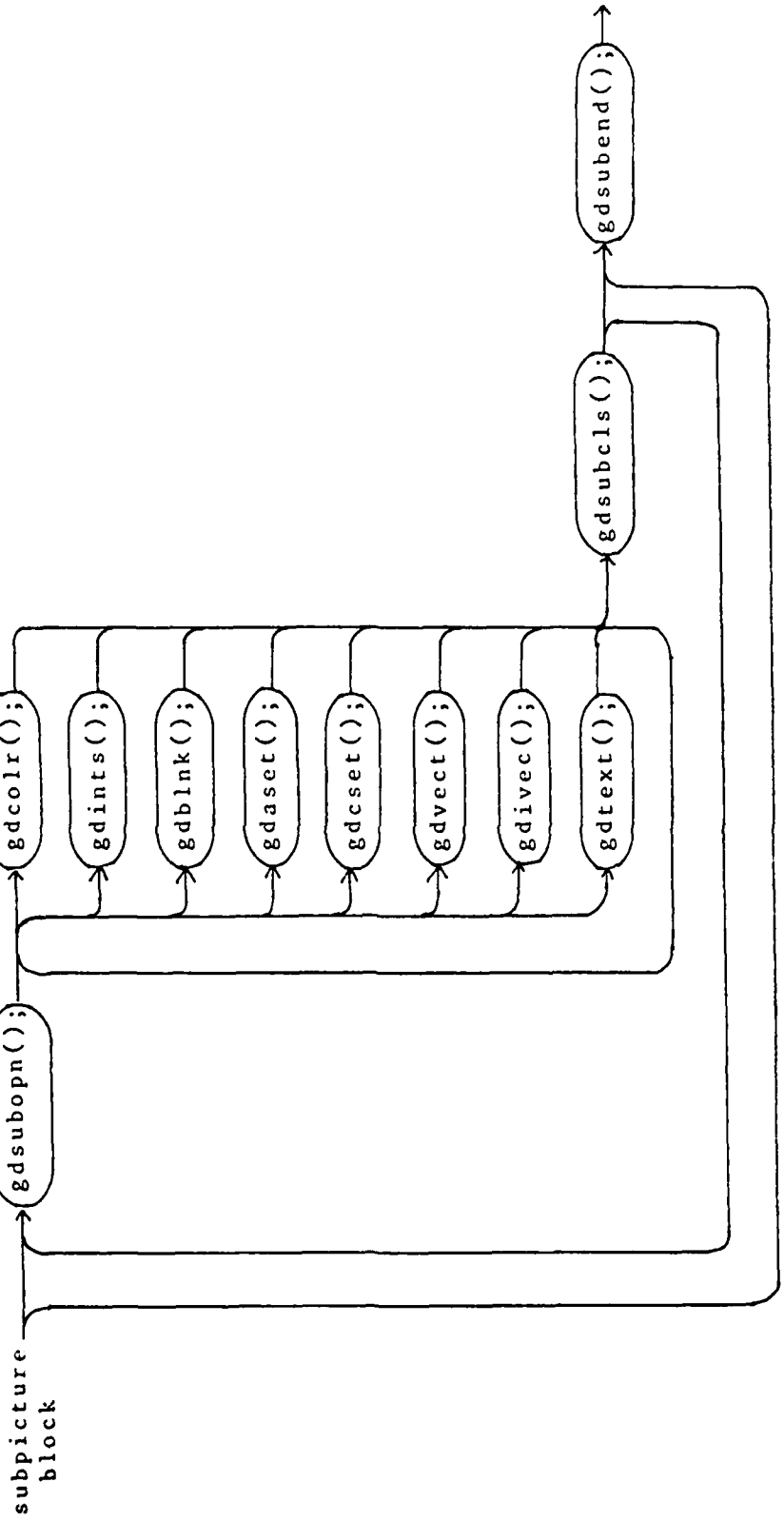
```
ininit(flag)

/* This routine does the set up to receive signals from "inread" to
     indicate there is something to read.                          */

int  *flag;

     {
     static inflag;       /* Static variables don't disappear when */
     static *flagptr;     /* the routine is exited. */

     if (inflag == 0)     /* Initially inflag = 0, subsequent calls
                             inflag = 1.                           */
          {
          inflag = 1;
          flagptr = flag;
          *flagptr = 0;
          }
     else
          {
          *flagptr = 1;        /* flag is set to 1 */
          }

     signal(16, &ininit);      /* Everytime there is an input interrupt
                                  call ininit.                     */
     }
```

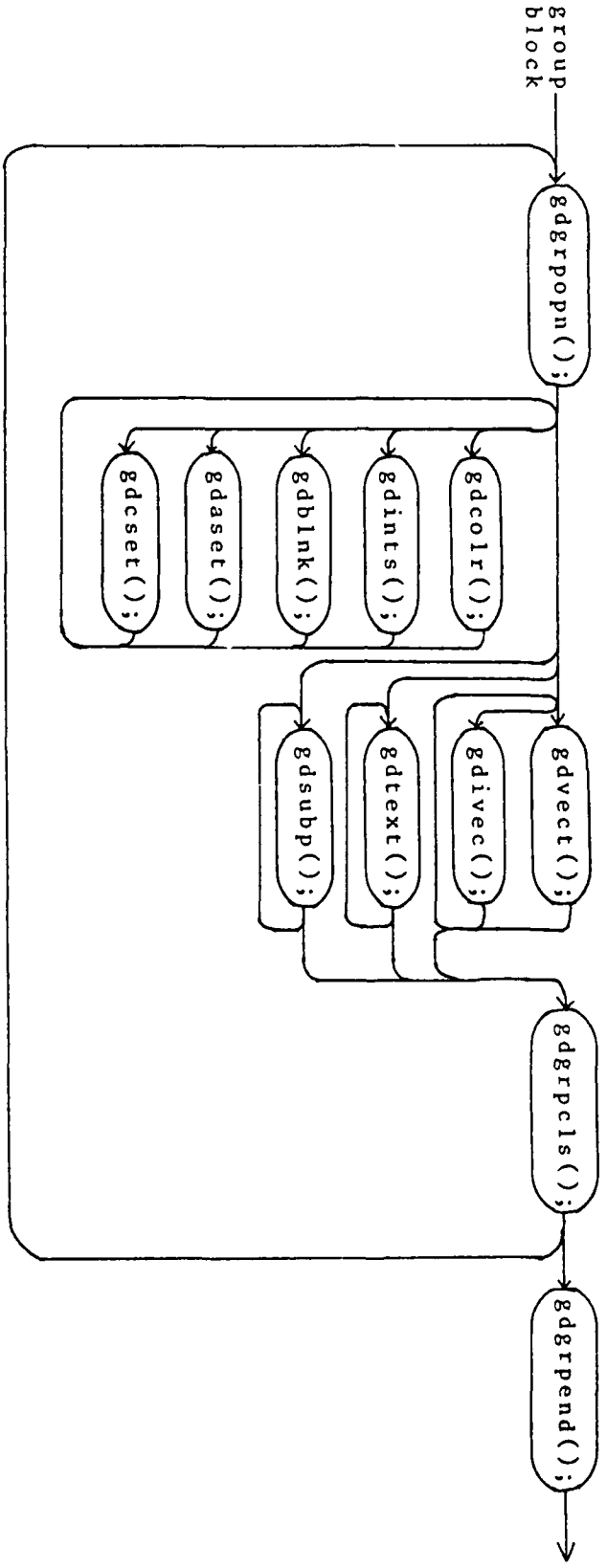
# APPENDIX 9

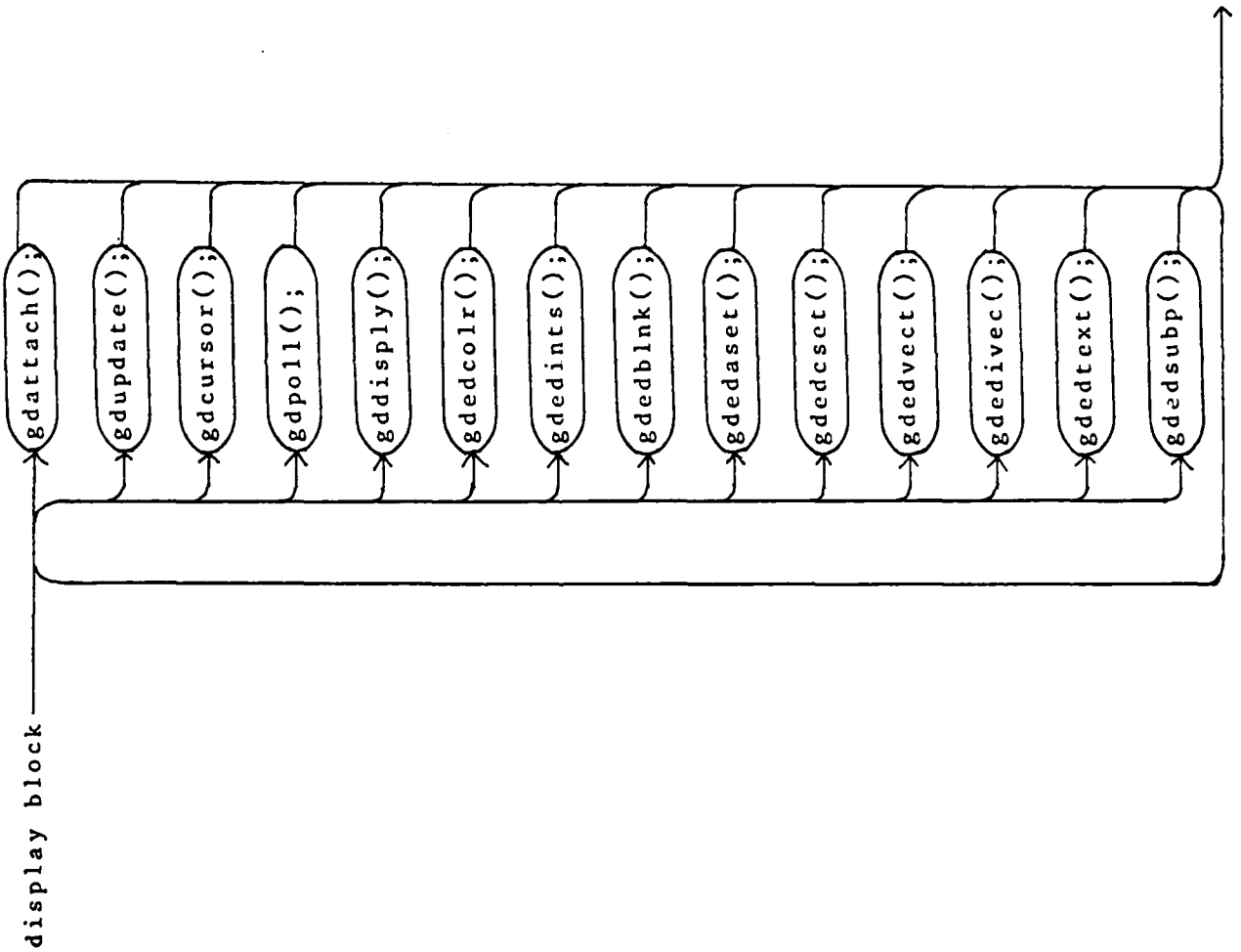
## THE SYNTAX DIAGRAMS FOR USING THE GRAPHICS DISPLAY LIBRARY ROUTINES

In any C routine there can be a block of code to create a picture on the display of the VGS/VDP or RGS/VDP. The C code can be interspersed throughout the block.




subpicture
block
gdsubopn();
gdcolr();
gdints();
gdblnk();
gdaset();
gdcset();
gdvect();
gdivec();
gdtext();
gdsubcls();
gdsubend();


ground
block
gdgrdopn();
gdcolr();
gdints();
gdblnk();
gdaset();
gdcset();
gdvect();
gdivec();
gdtext();
gdsubp();
gdgrdcls();
gdgrdend();


display block
gdattach();
gdupdate();
gdcursor();
gdpoll();
gddisply();
gdedcolr();
gdedints();
gdedblnk();
gdedaset();
gdedcset();
gdedvect();
gdedivec();
gdedtext();
gdedsubp();

UNCLASSIFIED

Security Classification

**DOCUMENT CONTROL DATA - R & D**

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1 ORIGINATING ACTIVITY
Defence Research Establishment Ottawa
National Defence Headquarters
Ottawa, Ontario K1A 0Z4

2a. DOCUMENT SECURITY CLASSIFICATION
Unclassified

2b GROUP

3. DOCUMENT TITLE
A General Area Air Traffic Controller Simulation
Using Colour Graphics

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Report

5. AUTHOR(S) (Last name, first name, middle initial)
Ford, Barbara J.

| 6. DOCUMENT DATE | 7a. TOTAL NO OF PAGES | 7b. NO. OF REFS |
|---|---|---|
| September 1983 | 149 | 27 |

| 8a. PROJECT OR GRANT NO. | 9a. ORIGINATOR'S DOCUMENT NUMBER(S) |
|---|---|
| 31B | DREO REPORT 890 |
| 8b. CONTRACT NO. | 9b. OTHER DOCUMENT NO.(S) (Any other numbers that may be assigned this document) |

10. DISTRIBUTION STATEMENT
Unlimited distribution

| 11. SUPPLEMENTARY NOTES | 12. SPONSORING ACTIVITY |
|---|---|
| | CRAD |

13. ABSTRACT

The increased speed of aircraft and a greater density of air traffic are overtaxing air traffic controllers. Automation of some portions of the controller's information and the use of colour to highlight dangerous situations will make the controller's work easier and possibly more accurate. One of the major aims of the study is to show that the use of different colours to indicate varying distances between aircraft is a definite improvement over a monochrome display. The other major aim is to study the advantages and disadvantages of vector display and raster display technologies when applied to the air traffic controller scenario. A new set of graphics display routines was developed to be used for the simulation. Much consideration was given to the best methods to optimize the display routines, with attention given to real-time constraints.

DSIS

77-065

UNCLASSIFIED
Security Classification

**KEY WORDS**

Secondary Display
Colour Graphics
Air Traffic Control
Programming Guide
Vector Refresh Display
Raster Refresh Display
Monochrome Display Comparison
Simulation
Interaction

**INSTRUCTIONS**

1. ORIGINATING ACTIVITY: Enter the name and address of the organization issuing the document.

2a. DOCUMENT SECURITY CLASSIFICATION: Enter the overall security classification of the document including special warning terms whenever applicable.

2b. GROUP: Enter security reclassification group number. The three groups are defined in Appendix 'M' of the DRB Security Regulations.

3. DOCUMENT TITLE: Enter the complete document title in all capital letters. Titles in all cases should be unclassified. If a sufficiently descriptive title cannot be selected without classification, show title classification with the usual one-capital-letter abbreviation in parentheses immediately following the title.

4. DESCRIPTIVE NOTES: Enter the category of document, e.g. technical report, technical note or technical letter. If appropriate, enter the type of document, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the document. Enter last name, first name, middle initial. If military, show rank. The name of the principal author is an absolute minimum requirement.

6. DOCUMENT DATE: Enter the date (month, year) of Establishment approval for publication of the document.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the document.

8a. PROJECT OR GRANT NUMBER: If appropriate, enter the applicable research and development project or grant number under which the document was written.

8b. CONTRACT NUMBER: If appropriate, enter the applicable number under which the document was written.

9a. ORIGINATOR'S DOCUMENT NUMBER(S): Enter the official document number by which the document will be identified and controlled by the originating activity. This number must be unique to this document.

9b. OTHER DOCUMENT NUMBER(S): If the document has been assigned any other document numbers (either by the originator or by the sponsor), also enter this number(s).

10. DISTRIBUTION STATEMENT: Enter any limitations on further dissemination of the document, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this document from their defence documentation center."

(2) "Announcement and dissemination of this document is not authorized without prior approval from originating activity."

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document, even though it may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall end with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (TS), (S), (C), (R), or (U).

The length of the abstract should be limited to 20 single-spaced standard typewritten lines; 7½ inches long.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a document and could be helpful in cataloging the document. Key words should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context.

# END

# FILMED

6-85

# DTIC